



Quadcopter Design Project

ME 445 FINAL REPORT

JAMES DRYDEN & RYAN BARBACCIA

Introduction

Our objective for this project was to design and build a quad-rotor aircraft (hereafter referred to as “quadcopter”) that was capable of stable flight with manual radio control. More specifically, our goal was to implement an Arduino microcontroller as the quadcopter’s flight controller, and program this Arduino with our own custom code. While there are many open-source programs that enable Arduino boards to be used as flight controllers, we wanted to develop the code ourselves in order to learn more about what makes quadcopter flight possible. The purpose of this report is to describe results of the project and the steps that were necessary to reach those results.

We would like to thank Mike Robinson for sharing his knowledge of mechatronics with us, and for all of the help in the lab that made this project possible. We would also like to thank Dr. Sommer for teaching us so much throughout the semester.

Table of Contents

Introduction	1
1. Quadcopter Essentials	3
1.1 Quadcopter Frame	3
1.2 Motors and Propellers	3
1.3 Speed Controllers	4
1.4 Battery	4
1.5 Radio Receiver	4
1.6 Flight Controller	5
1.7 Attitude Sensor	5
2. Dynamic System Control	6
2.1 Determining Quadcopter Orientation	6
2.2 PID Control	7
3. Implementation	8
3.1 Receiver	8
3.1.1 Interrupts	8
3.1.2 PinChangeInt Library	9
3.2 Sensors	9
3.2.1 I ² C Communication	9
3.2.2 Sensor Configuration	10
3.2.3 State Estimation	10
3.3 Motors/ESCs	11
3.3 Vibration Control	12
3.4 Arduino	13
4. Hardware Diagram	14
5. Code Workflow	15
5.1 Setup/Initialization Workflow	15
5.2 Main Loop Workflow	16
6. Testing Procedures & Results	19
6.1 Dynamic Attitude Sensor Testing	19
6.2 Motor Thrust Testing	21
6.3 Stabilization Testing & PID Tuning	23
6.4 In-Flight PID Tuning	24
Project Code	25
Bill of Materials	26
References	27

1. Quadcopter Essentials

Every radio controlled (RC) quadcopter requires, at minimum, the following components: a frame, motors with propellers, motor speed controllers, a battery, a radio receiver, a flight controller, and an attitude sensor. This section will discuss the function of each component, why it is necessary, and what considerations to make when selecting that component.

1.1 Quadcopter Frame

The frame of the quadcopter provides the physical structure for the entire aircraft. It joins the motors to the rest of the aircraft and houses all of the other components. The frame must be large enough to allow all four propellers to spin without collision, but must not be too large and therefore too heavy for the motors.

For our quadcopter we chose a Hobbyking SK450 frame as seen in Figure 1, which measures at 450mm across opposite motors.



Figure 1: Hobbyking SK450 Frame

1.2 Motors and Propellers

The motors spin the propellers to provide the quadcopter with lifting thrust. Quadcopters almost exclusively use brushless DC motors, as they provide thrust-to-weight ratios superior to brushed DC motors. However, they require more complex speed controllers.

Hobby motors are typically given two ratings: Kv ratings and current ratings. The Kv rating indicates how fast the motor will spin (RPM) for 1 V of applied voltage. The current rating indicates the max current that the motor may safely draw. For our project, we selected 1000Kv, 15A max NTM motors from Hobbyking (seen in Figure 2).



Figure 2: NTM 100Kv Motor

Propellers come in many sizes and materials. They are measured by their diameter and their pitch, in the format (diameter) x (pitch). Pitch is a measurement of how far a propeller will “travel” in one revolution. Prop selection is important to yield appropriate thrust while not overheating the motors. For our project, we selected 9x4.7 carbon fiber props (Figure 3), which yield 1.4lbs of max thrust while drawing 10.2A.



Figure 3: 9x4.7 Carbon Fiber Props

With four motors, the max thrust for the quad is approximately 5.5 lbs. Our quad has an all-up-weight of 2.0 lbs (925 g), resulting in an overall thrust-to-weight ratio of 2.75. This allows the quad to hover just below half-throttle.

1.3 Speed Controllers

Every motor needs an individual electronic speed controller (ESC for short). These speed controllers accept commands in the form of PWM signals and output the appropriate motor speed accordingly. Every ESC has a current rating, which indicated the maximum current that it may provide the motor without overheating. Appropriate ESCs must be chosen to ensure that they can provide enough current for the motors. We selected 20A Afro ESC for our project (Figure 4), as they are well reviewed for use with quadcopters and have a sufficient current rating.



Figure 4: Afro ESC 20A

1.4 Battery

The battery provides electrical power to the motors and all electronic components of the aircraft. Lithium Polymer (LiPo) batteries are used almost exclusively, because they have high specific energy. Hobby LiPo batteries have a capacity rating and discharge rating. The capacity rating, in milliamp-hours (mAh) indicates how much current the battery may output for one hour. Discharge rating, indicated by the letter "C", shows how fast the battery may be safely discharged. To determine max allowed current, multiply the C value with the capacity. For this project, we selected Turnigy 2200mAh 20C LiPo batteries, seen in Figure 5.



Figure 5: Turnigy 2200mAh 20C LiPo

1.5 Radio Receiver

The radio receiver (Rx) receives radio signals from an RC transmitter and converts them into control signals for each control channel (throttle, yaw, roll & pitch). Modern RC receivers operate on a 2.4 GHz radio frequency, while older Rx units often used 72 MHz frequencies. Rx units may have as few as 4 channels, but many have more channels for additional control options. We selected a Hobbyking OrangeRx 6 Channel Receiver for this project, seen in Figure 6.



Figure 6: OrangeRx 6Ch Receiver

1.6 Flight Controller

The flight controller is the “brain” of the quadcopter, and performs the necessary operations to keep the quadcopter stable and controllable. It accepts user control commands from the Rx, combines them with readings from the attitude sensor(s), and calculates the necessary motor output.

For most hobby quadcopters, one would select a purpose-made flight controller board. These boards often have integrated attitude sensors, and provide well-tested flight control software. For our project however, we used an Arduino Pro Mini (seen in Figure 7) as the flight controller, as we intended to program the flight control software ourselves.

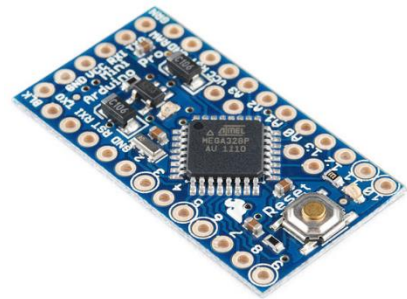


Figure 7: Arduino Pro Mini

1.7 Attitude Sensor

The attitude sensor provides the flight controller with readings of the quadcopter’s orientation in space. At minimum this requires a gyroscope, but most quadcopters also incorporate an accelerometer. For a self-stabilizing quadcopter (such as ours), an accelerometer is required.

For a quadcopter application, a 6-axis inertial measurement unit (IMU) is desired, consisting of a gyroscope and accelerometer on the same board. We selected a [Sparkfun 9DOF Sensor Stick](#), seen in Figure 8. This board includes an ADXL345 accelerometer, an ITG-3200 gyroscope, and an HMC3885L magnetometer.

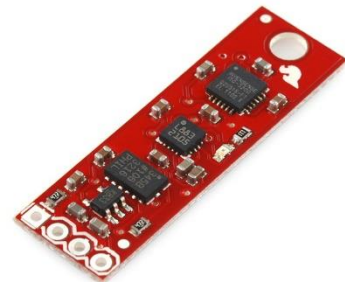


Figure 8: Sparkfun 9DOF Sensor Stick

Component images courtesy of HobbyKing.com Sparkfun.com

2. Dynamic System Control

A quadcopter is inherently a very unstable system. Anyone attempting to control a quadcopter with manual inputs only and no attitude sensor integration would quickly find that balancing the aircraft is very nearly impossible. So, to make the quadcopter a stable system, it is important to integrate an attitude sensor and a set of dynamic system controllers. For our quadcopter, we chose to use PID controllers due to their relative simplicity and good performance.

2.1 Determining Quadcopter Orientation

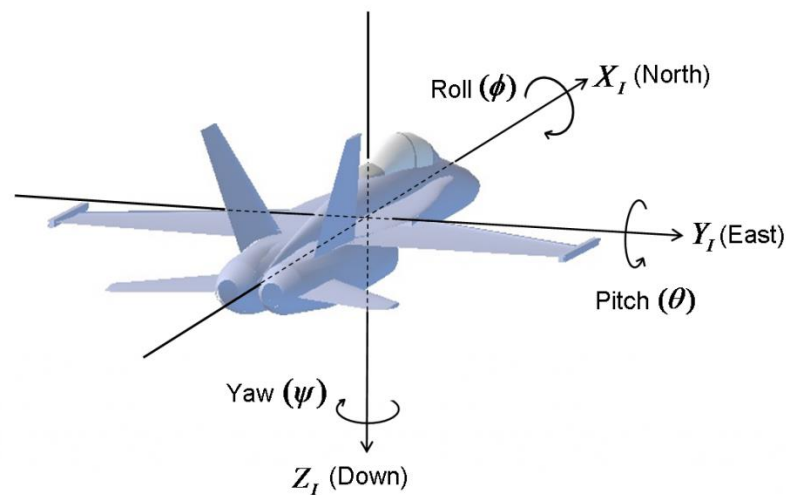


Figure 9: Inertial Frame of a Free Body (from chrobotics.com)

In order to stabilize the quadcopter, it is first crucial to determine the aircraft orientation (also called attitude) relative to the fixed inertial frame of the earth. This inertial frame is shown in Figure 9, and consists of 3 orthogonal axes (North, East, & Down) and the rotations about these axes (Roll, Pitch, and Yaw).

In order to attain stable flight, the roll and pitch axes must first be stabilized. If these axes are not properly controlled, the quadcopter will immediately tip over and be unable to fly. The roll and pitch attitudes of the aircraft are determined using the attitude sensor. In the case of our project, this was done using the gyroscope and accelerometer in conjunction, in a manner which will be discussed in more detail in subsection 3.2 Sensors.

The yaw axis must also be relatively stable for the quadcopter to be controllable, but is less critical. Slight drift in the yaw axis is easily counteracted using the radio controller, and usually will not result in a loss of control. Using only an accelerometer and gyroscope (as was done in this project), the absolute yaw orientation is in fact not measurable. Only the *change* in yaw orientation is measurable by using the gyroscope, but this proved to be sufficient to enable control of the quadcopter.

2.2 PID Control

After determining the attitude of the aircraft, it is necessary to implement a dynamic system controller to stabilize the quadcopter at the desired attitude (often simply level). One of the most effective methods of doing so is to implement a proportional, integral and derivative (PID) controller.

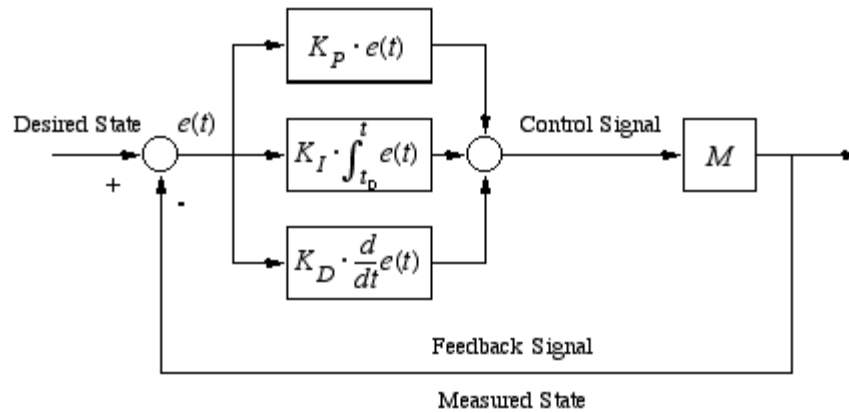


Figure 10: Standard PID Block Diagram

PID control is shown in block diagram form in Figure 10 and is performed in the following steps:

1. The error $e(t)$ is calculated as *Set point* – *Measured State*
2. The proportional term P is calculated as $K_p \cdot e(t)$
3. The integral term I is calculated as $K_I \cdot (\text{time integral of } e(t))$
4. The derivative term D is calculated as $K_D \cdot (\text{time derivative of } e(t))$
5. The 3 terms are summed to produce the controller output, $u(t) = P + I + D$

In order to stabilize the quadcopter, a separate PID controller was implemented for the roll, pitch, and yaw axes.

As mentioned in the previous section, an absolute measurement of the yaw axis is not available using only an accelerometer and gyroscope. So to stabilize the quadcopter about the yaw axis, the PID controller was implemented to control the rate of rotation about the yaw axis.

PID control only produces desired performance when the three control gains, $\{K_p, K_I, K_D\}$ are properly selected. The process of selecting these parameters is referred to as “tuning” the PID.

3. Implementation

While the concepts of how a quadcopter operates are simple, implementing each subsystem requires quite a bit of attention to detail in order for the aircraft to function properly. This section will discuss the details of how each system works and what was necessary to implement it.

3.1 Receiver

The RC receiver accepts radio signals from an RC transmitter and translates it into separate channels of control. The receiver in our quadcopter is capable of outputting 6 channels of control, including throttle, yaw, roll, pitch, and 2 auxiliary channels (controlled by toggle switches on the transmitter).

RC signals are a form of specialized PWM, in which the length of the HIGH pulse contains the output information, as seen in Figure 11. Each HIGH pulse varies from approximately 1 ms to 2 ms, with a period of 20 ms.

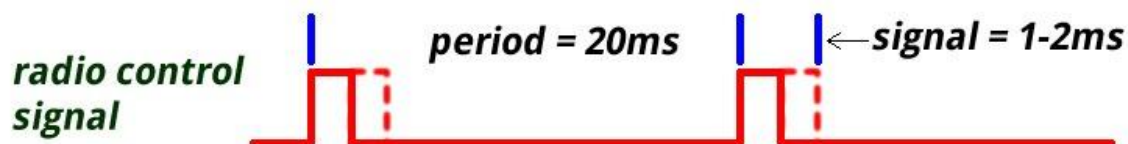


Figure 11: Standard Radio Control Signal

In order to read this signal into the Arduino flight controller, the Arduino needs to measure the length in microseconds of the HIGH pulse. The simplest way to do so is to use the `pulseIn()` function, which measures pulse lengths on a pin. However, this method is not suitable for a flight controller because the function blocks the rest of the program from running while it waits for a pulse.

3.1.1 Interrupts

A much better way to measure RC signals is to make use of interrupts. Interrupts are functions that are triggered by changes on a specified I/O pin on the Arduino, and may interrupt the main program to execute. An example interrupt function to measure an RC signal may be seen in Figure 12.

To measure multiple channels of RC input, the Arduino must have an individual interrupt for each channel. Because there are only 2 external interrupts available on the Arduino Pro Mini, we must use a library called *PinChangeInt*.

```
void interruptFuncChan0()
// Interrupt function to measure RC pulses
{
    // if pin is high, new pulse. Record time
    if (digitalRead(RX_PIN_CH0) == HIGH)
    {
        rxStart[0] = micros();
    }
    // if pin is low, end of pulse. Calculate
    // pulse length and add channel flag. Also
    // reset channel start.
    else
    {
        rxValShared[0] = micros() - rxStart[0];
        rxStart[0] = 0;
        updateFlagsShared |= RX_FLAG_CH0;
    }
}
```

Figure 12: Example Interrupt Function

3.1.2 PinChangeInt Library

The PinChangeInt Library is a 3rd party Arduino library that allows you to take advantage of “pin change” interrupts, which are available on each IO pin of the Arduino. The library provides nearly identical performance to standard external interrupts, but allows for many more separate interrupt functions to be used. This is ideal for reading 4 or more channels of RC input.

For more info on the PinChangeInt Library, visit <http://playground.arduino.cc/Main/PinChangeInt>.

3.2 Sensors

For our project we selected the [9DOF Sensor Stick from SparkFun](#). This board contains an ADXL345 Accelerometer, ITG-3200 Gyroscope, and HMC5883L Magnetometer. All three sensors are communicated with by I²C serial communication. When configured correctly, they provide reliable inertial measurements that are combined using State Estimation to determine the aircraft attitude.

3.2.1 I²C Communication

To connect the sensor board to the Arduino, only two pins are required for SCL and SDA on the I²C bus. A wiring diagram for I²C on the Arduino Pro Mini may be seen in **Error! Reference source not found..**

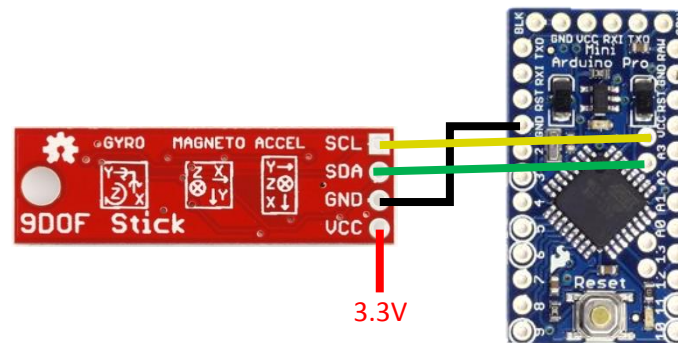


Figure 13: I²C Wiring Diagram for Arduino Pro Mini

During I²C communication, the master device (Arduino) dictates the clock speed by pulling SCL HIGH or LOW at the desired speed. Data may be sent in either direction on the SDA line in sync with SCL speed. Each I²C slave device on the bus has its own unique I²C address, in the form of an 8-bit number, which allows the master device individually address each slave device. These addresses may be found in the data sheets for each device.

I²C communication is made simpler using the Arduino Wire library, which operates by default at 100 kHz. However, if the slave devices support “I²C Fast Mode,” this clock speed may be increased to 400 kHz, providing a significant decrease in the time needed to read each sensor. Because the ADXL345 and ITG-3200 both support 400 kHz operation in I²C Fast Mode, our program takes advantage of this ability.

The I²C clock speed is determined by the TWBR register in the on-board TWI module of the ATmega328p. The Wire library automatically sets this register to 72, which results in 100 kHz operation.

To switch I²C communications to 400 kHz, manually set the value of the TWBR register to 12 by including the command `TWBR = 12;` after initializing the Wire library.

For more information on the Wire library and the ATmega328 TWI module, visit <http://playground.arduino.cc/Code/ATMELTWI>.

3.2.2 Sensor Configuration

During the setup of the Arduino program, the program must initialize the sensors by writing configuration parameters to corresponding registers on each I²C device. These configuration parameters determine sensor operation behavior such as update rate, resolution, and other miscellaneous options. In order to have the sensors return reliable values, it is very important that these configurations be made.

The ADXL345 Accelerometer is capable of updating at 100 Hz and can measure accelerations of +/- 16g. For our use, we configured the accelerometer at a rate of 50 Hz, which yields more reliable readings. The resolution is also user-selectable, and for our project, we selected “Full-Resolution Mode,” which dynamically changes resolution to maintain a scale of 4 mg/LSB. This resolution mode was determined to yield the most reliable acceleration readings.

The ITG-3200 Gyroscope is capable of being sampled at 8 kHz. It contains a built in digital low pass filter with selectable cut-off frequencies. For best performance we configured the sampling rate to be at its maximum (8 kHz) and set the low pass filter cutoff frequency to 256 Hz (its highest value). Although we are not able to sample the gyroscope at anywhere near 8 kHz, this setting ensures that a new value is available every time the Arduino requests it.

While the sensor board contains a magnetometer, it has not been implemented into our flight controller code and is currently unused.

3.2.3 State Estimation

While the accelerometer gives us an absolute measurement of the quadcopter attitude, accelerometers are very prone to noise. The motors on the quadcopter produce a lot of vibration, introducing significant noise into the accelerometer reading. The gyroscope is much less effected by vibration, but it only gives us angular rotation rates. By integrating the gyroscope readings, it is possible to estimate the attitude, but this reading is prone to drifting over time.

In order to get the best estimate of quadcopter attitude, it is important to combine the accelerometer and gyroscope readings using “Sensor Fusion” or “State Estimation.” To do this in our flight controller we implemented a state estimation algorithm adapted from Mike Robinson’s state estimation presentation given in class. This algorithm may be seen in pseudo-code in Figure 15, and in block diagram form in Figure 14.

In the algorithm, the influence of the accelerometer is determined by the accelerometer gain. The higher the value of the gain, the more weight the accelerometer reading has in the estimation. For greater noise immunity, this gain should be low, to favor the gyroscope. If the value is too low, however, the estimated angle may drift from the actual angle.

Using the algorithm we were able to obtain very reliable attitude measurements even in a high vibration environment when the motors are spinning. Being able to accurately measure the quadcopter attitude is absolutely critical to the successful operation of the quadcopter.

```

accelerometer angle = atan2(accel_y, accel_x)
angle error = estimated angle - accelerometer angle
estimated angle = estimated angle +
    (gyroscope reading - angle error * gain) * dt
    
```

Figure 15: State estimation pseudo-code

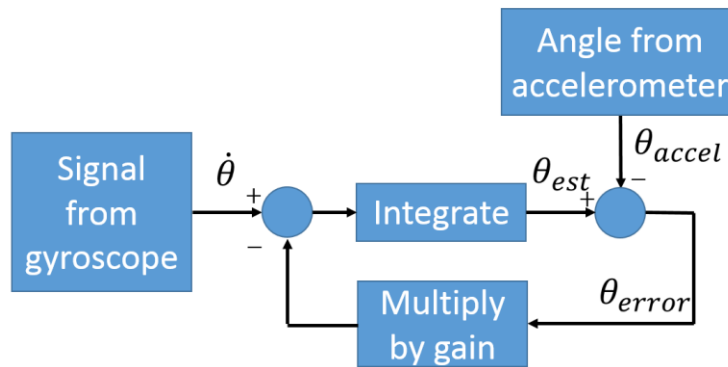


Figure 14: State estimation block diagram.

3.3 Motors/ESCs

To control the speed of the motors on the quadcopter, the flight controller sends signals to each speed controller (ESC), and the ESC spins the motors according to that signal. The signal that is sent to the ESCs is similar to the RC receiver signal. The motor speed is determined by the length of the high pulse, but the period of the signal does not have to be 20 ms. Some ESCs are compatible with update rates up to 490 Hz, including the ESCs that we chose for this project. The faster the update rate of the ESCs, the more precisely the speed of the motors can be controlled by the flight controller.

If fast update rate is not critical, the control signals may be sent to the ESCs using the servo library. Servo signals follow the same format as ESC signals, but are limited to an update rate of 50 Hz. One advantage of using the servo library, is that any IO pin on the Arduino may be used to send the signal.

Because we wanted the fastest possible update rate, we used the hardware PWM capabilities of the Arduino Pro Mini. This Arduino has six PWM capable IO pins, whose frequencies are controlled by three different timers: timer 0, timer 1, and timer 2. The pin assignment and default frequency for each timer is shown in Table 1.

Table 1: Arduino PWM Timers

	I/O Pins	Freq.
Timer 0	5, 6	976 Hz
Timer 1	9, 10	490 Hz
Timer 2	3, 11	490 Hz

Timer 0 operates at too fast of a frequency for ESC control. While this frequency may be changed, timer 0 is used for the millis and micros functions on the Arduino, and changing timer 0 would disrupt their functionality. Because we only needed four outputs, we used pins 9, 10, 3, and 11 to send signals to the ESCs.

The analogWrite function may be used to output PWM to the speed controllers, and is the simplest way to do so. However, for the sake of efficiency, the PWM outputs may be controlled by directly accessing the PWM hardware registers. These PWM output values must be 8-bit numbers representing the HIGH pulse length. In our program, motor speed was calculated in values between 1000 and 2000. Therefore to write these values to the PWM registers, they had to be divided by eight, to scale the numbers to 125 to 250. These values correspond to the correct PWM duty cycle. Because the numbers are being divided by eight, the division may be performed using bit shifting, which is performed faster by the Arduino.

3.3 Vibration Control

Excessive vibrations on a quadcopter may result in inaccurate attitude measurements, even when using sensor fusion to combine the gyroscope and accelerometer readings. While the state estimation algorithm used in this project is effective, it is not immune to noise due to vibrations. Therefore, these vibrations need to be minimized as much as possible, and the sensors should be isolated from any remaining vibrations.

The most important step in minimizing vibrations is balancing the quadcopter's propellers. This involves placing the props on a balancing stand which allows the props to rotate freely with nearly zero friction and adding or removing material from the side that tends to fall down. For our project, we used a prop balancer by DuBro, seen in Figure 16. This balancing should be performed along both blades, and across the hub. A well-balanced propeller should be able to come to a halt at any orientation on the balancing stand.



Figure 16: DuBro Propeller Balancer

After reducing vibrations as much as possible by balancing the propellers, we isolated the sensor board from the remaining vibrations using a vibration-absorbing mount. To do this, we used foam ear plugs, similar to those seen in Figure 17. The ear plugs hold the top plate of the frame, including the flight controller, sensors, and RC receiver, to the rest of the quad. The ear plugs are designed for vibration absorption (in the form of sound), making them ideal for this application. The setup may be seen in Figure 18.



Figure 17: Foam Ear Plugs Used for Vibration Absorption

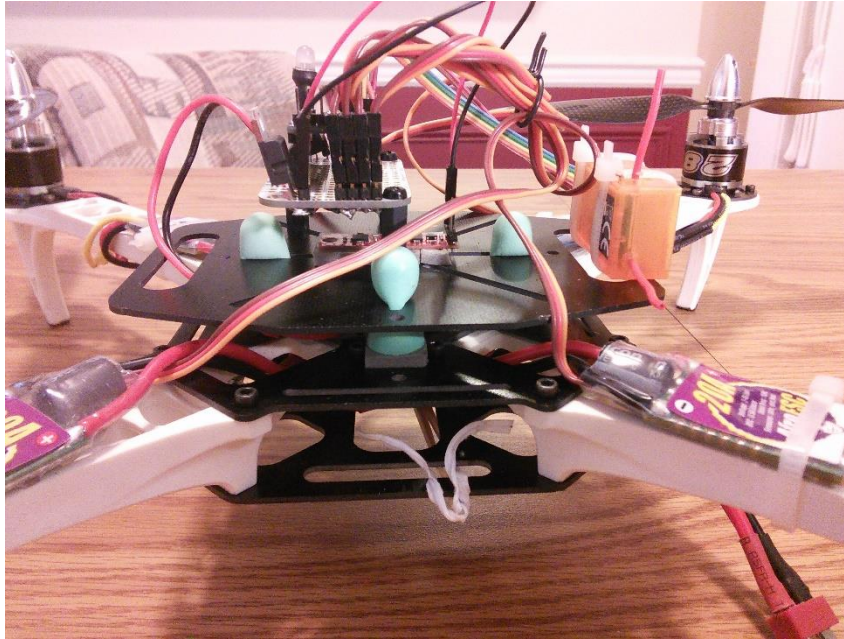


Figure 18: Vibration Absorption Mount on Quadcopter

3.4 Arduino

We selected the Arduino Pro Mini 328p as the flight controller for our quadcopter. Our code was initially developed for the Arduino UNO, but to conserve size and weight, the mini was selected instead. Because the mini uses the same Atmel ATmega328p micro-processor, the code was compatible. The pro mini has just as many IO pins as the UNO, but lacks an on board USB port. This conserves space, size, and cost, but means that programming the board requires an external FTDI board. The pro mini also operates at the same 5V and 16 MHz as the UNO, meaning that it is equally capable of performing the necessary operations as a flight controller.

To connect the RC receiver, sensors, and speed controllers to the Arduino, we fabricated a breakout board (seen in Figure 19) to hold the Arduino and provide reliable connections to each of the peripherals. The board was constructed using a prototyping perf-board. In the future a more permanent solution would be to create a PCB to hold the Arduino and the necessary connectors.

Power is supplied directly to the Arduino from the three cell LiPo battery via the on-board voltage regulator. The nominal voltage of the battery is 11.1 V, but at full charge the actual voltage may be as high as 12.6 V. While this is higher than the recommended 12 V for the regulator, it is below the absolute rated maximum of 16 V and has been tested without issue.

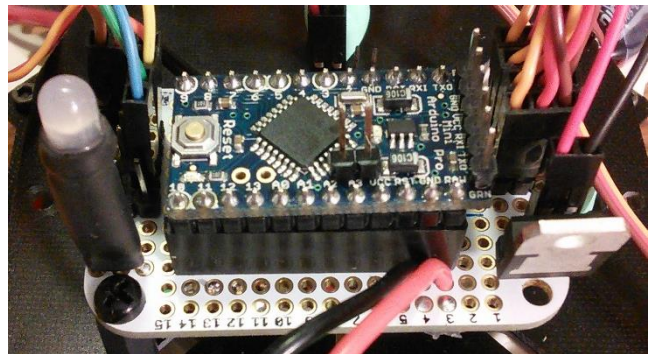
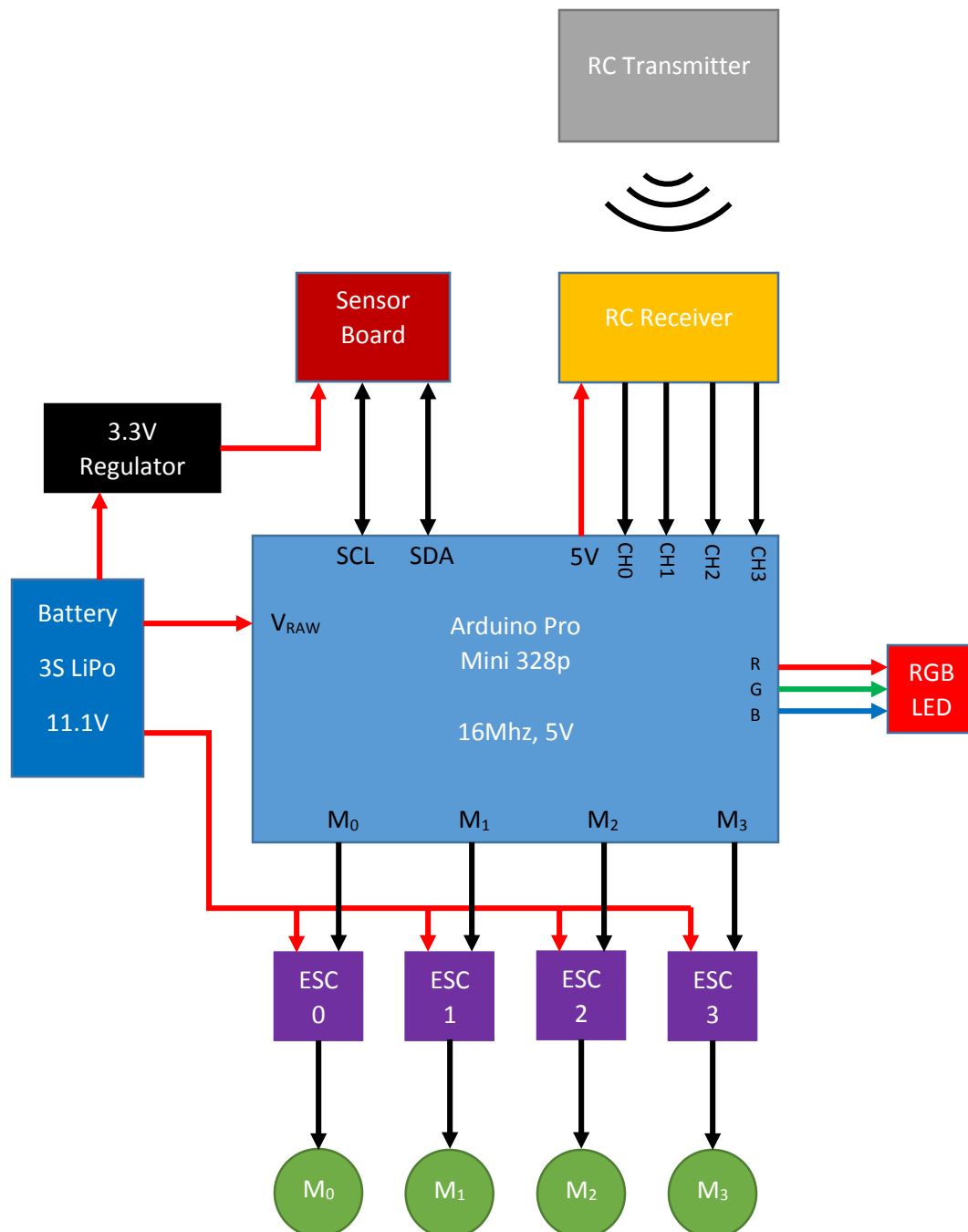


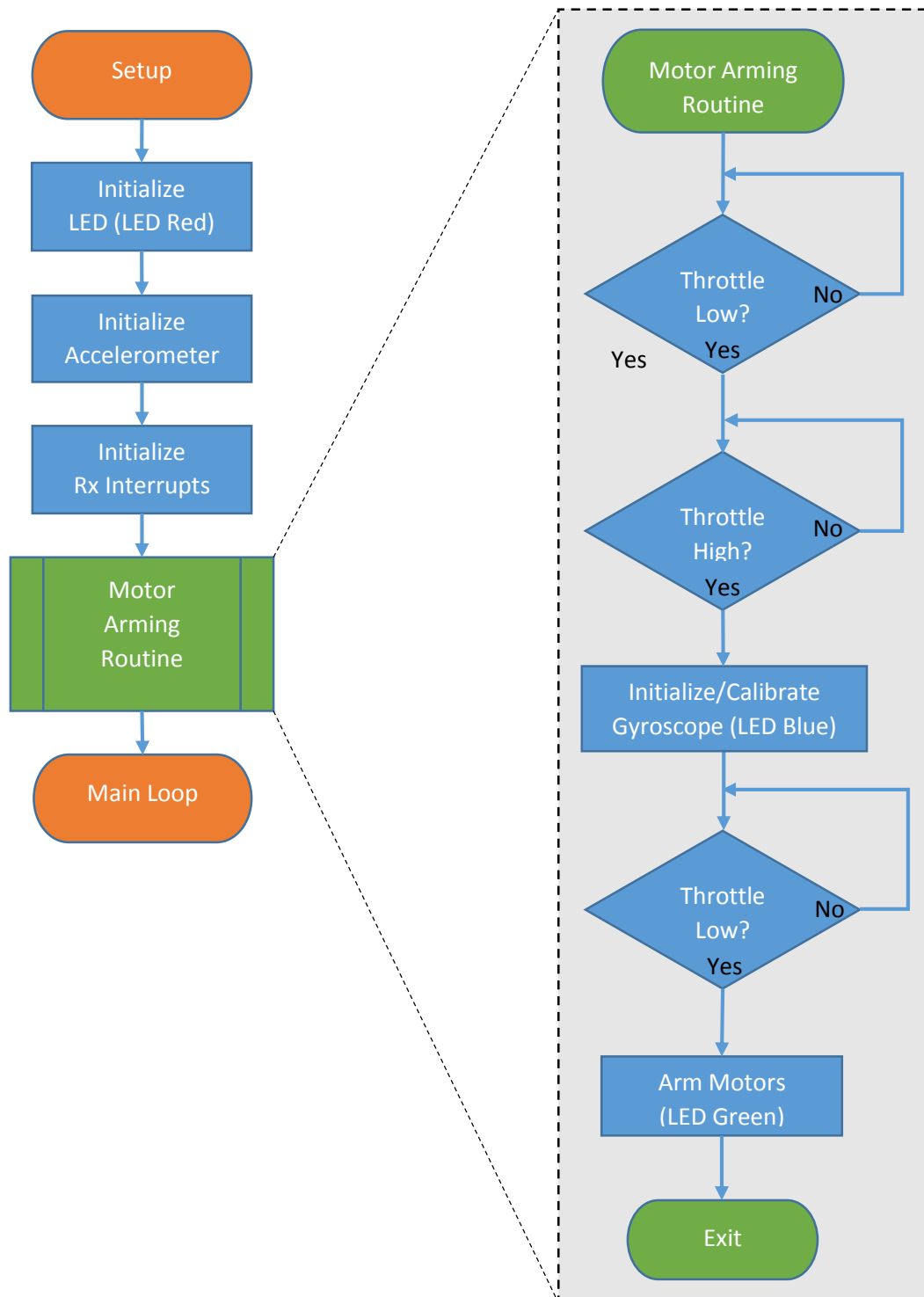
Figure 19: Breakout board for Arduino Pro Mini

4. Hardware Diagram

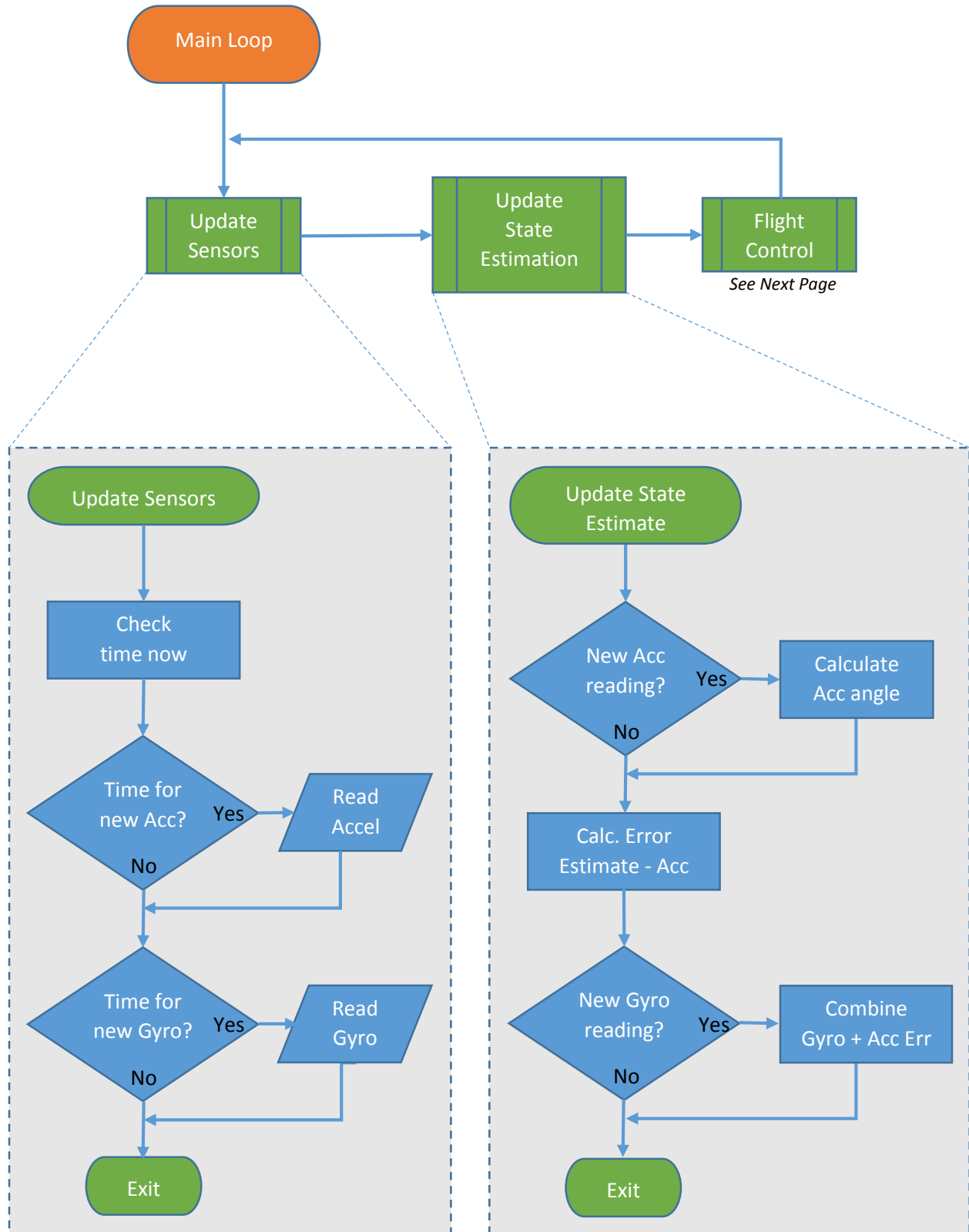


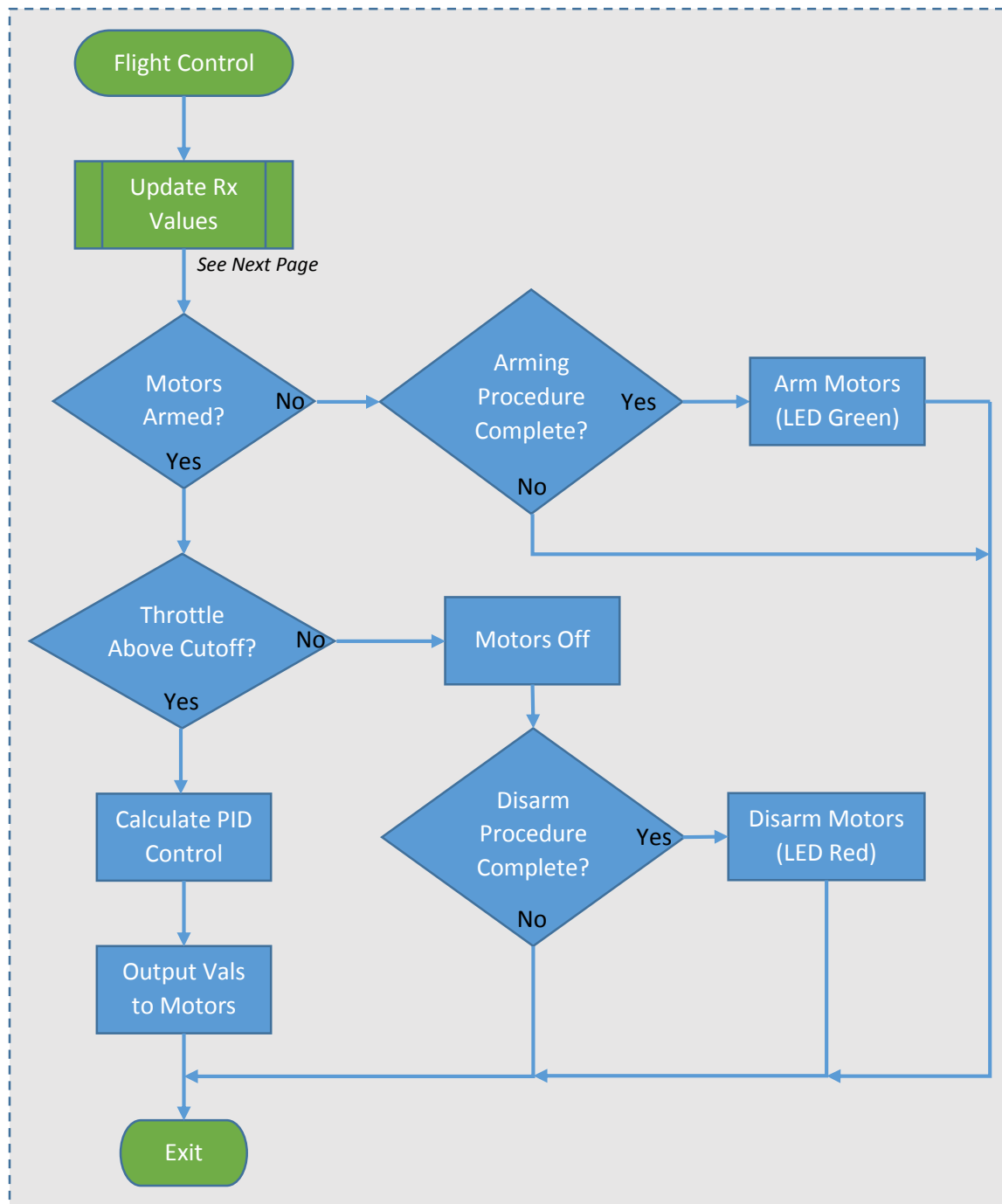
5. Code Workflow

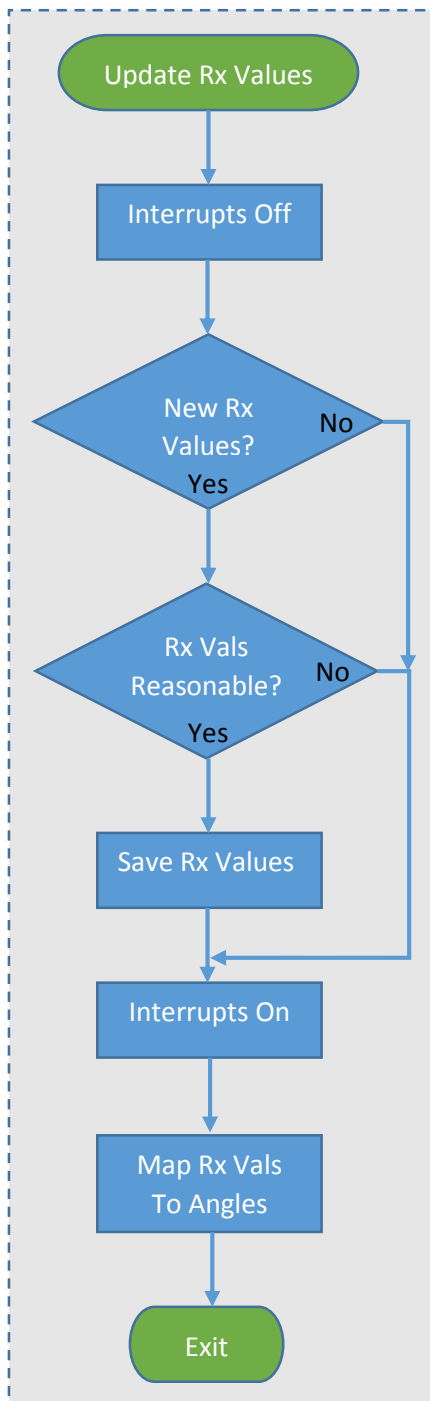
5.1 Setup/Initialization Workflow



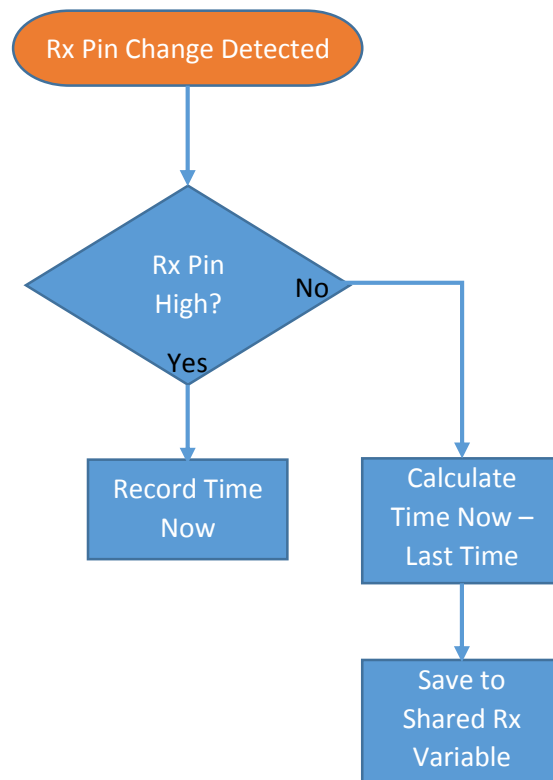
5.2 Main Loop Workflow







Interrupt Function



6. Testing Procedures & Results

Our project required significant testing at several stages throughout the project's duration. It was important that we tested the performance of critical systems as we developed them. This section covers the main tests that we performed during the development of the quadcopter.

6.1 Dynamic Attitude Sensor Testing

The first stage of our project was to obtain accurate attitude measurements using the ADXL345 and ITG-3200 and the state estimation algorithm. We first tested the attitude readings statically, orienting the sensor board at a known attitude and comparing the estimated angle with that known angle. Once satisfied, we moved on to a dynamic test of the attitude sensor's performance.

To test the sensor dynamically, we attached the sensor board to a Lego motor, which contains a built-in rotation encoder with a resolution of 1 degree/step, seen in Figure 20. We interfaced the motor's encoder with an Arduino UNO that was also taking sensor readings and performing the state estimation algorithm. Using this setup, we were able to collect simultaneous angle measurements using the encoder and the sensor board. This allowed us to compare the angle outputs of each while manually tilting the sensor about one axis.

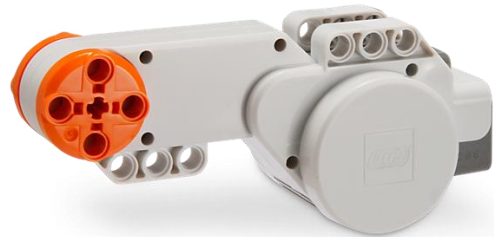


Figure 20: Lego Motor with Built-in Encoder

We printed out angle readings from both devices to the serial monitor on the Arduino IDE, and copied the data into excel. After shifting the beginning encoder angle to account for the starting angle not being level, we plotted the angle reading vs. time for each device. We repeated the experiment with several accelerometer gain settings, and saw consistently accurate performance from the sensor board.

The experiment was most useful when we used the vibration motor on a cell phone to introduce external vibrations to the system. As may be seen in Figure 22 on the next page, angle readings taken using the accelerometer only show significant noise. Using state estimation at a gain of 4 effectively eliminated this noise, as may be seen in Figure 21.

As we expected, the noise returned somewhat with state estimation at higher accelerometer gains. This may be seen in Figure 23, which shows the noise when the accelerometer gain is set to 15.

Our conclusion from this test was that state estimation with a low accelerometer gain is the most appropriate method of estimating the quadcopter attitude in a high-vibration environment.

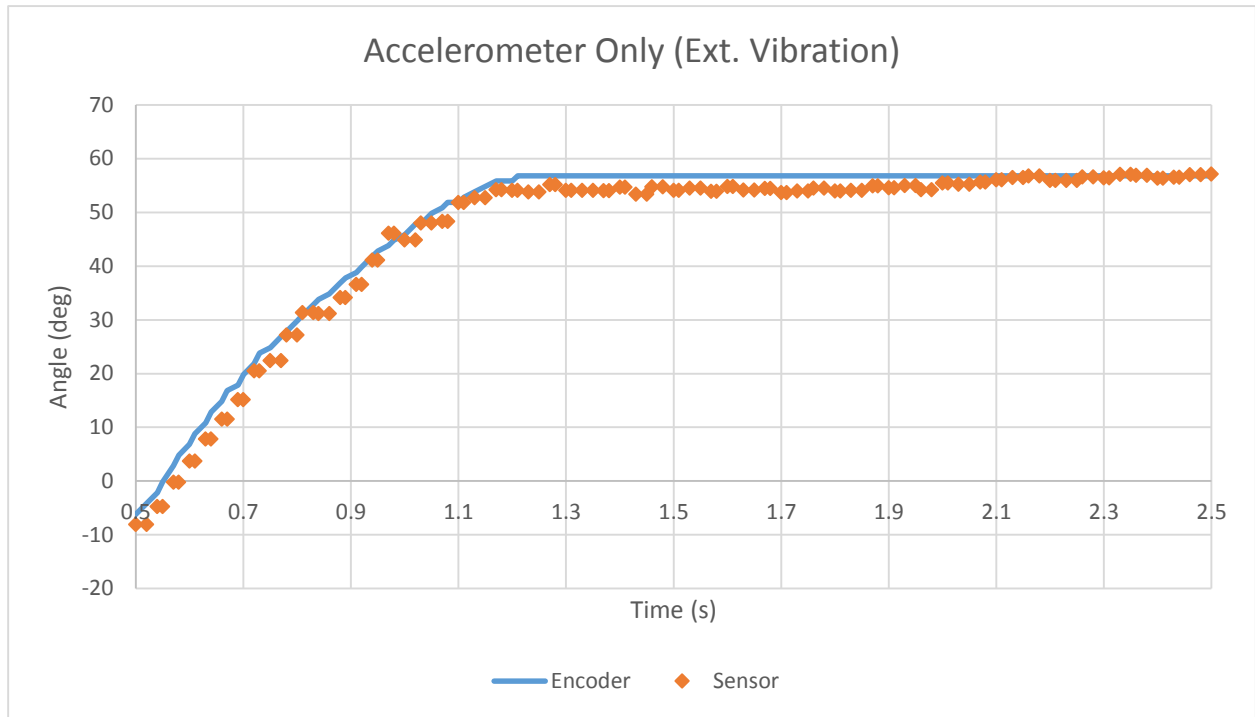


Figure 22: Dynamic Angle Testing: Accelerometer Only with External Vibrations

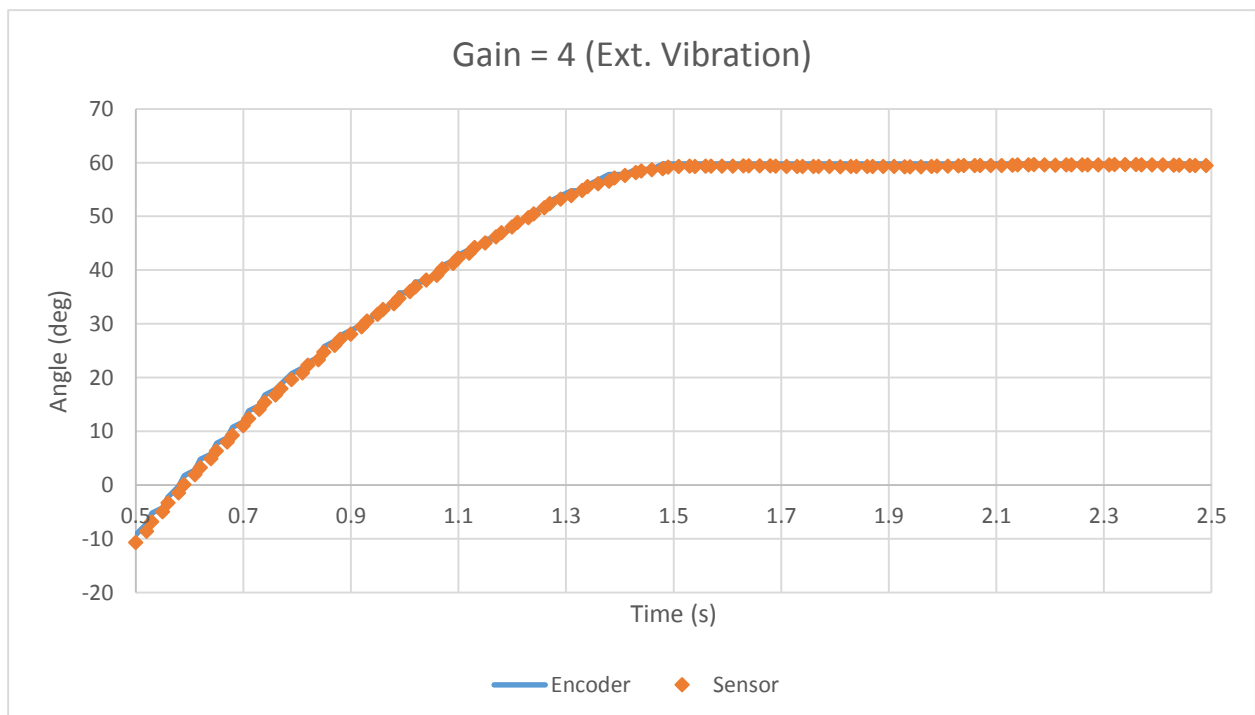


Figure 21: Dynamic Angle Testing: Gain=4 with External Vibrations

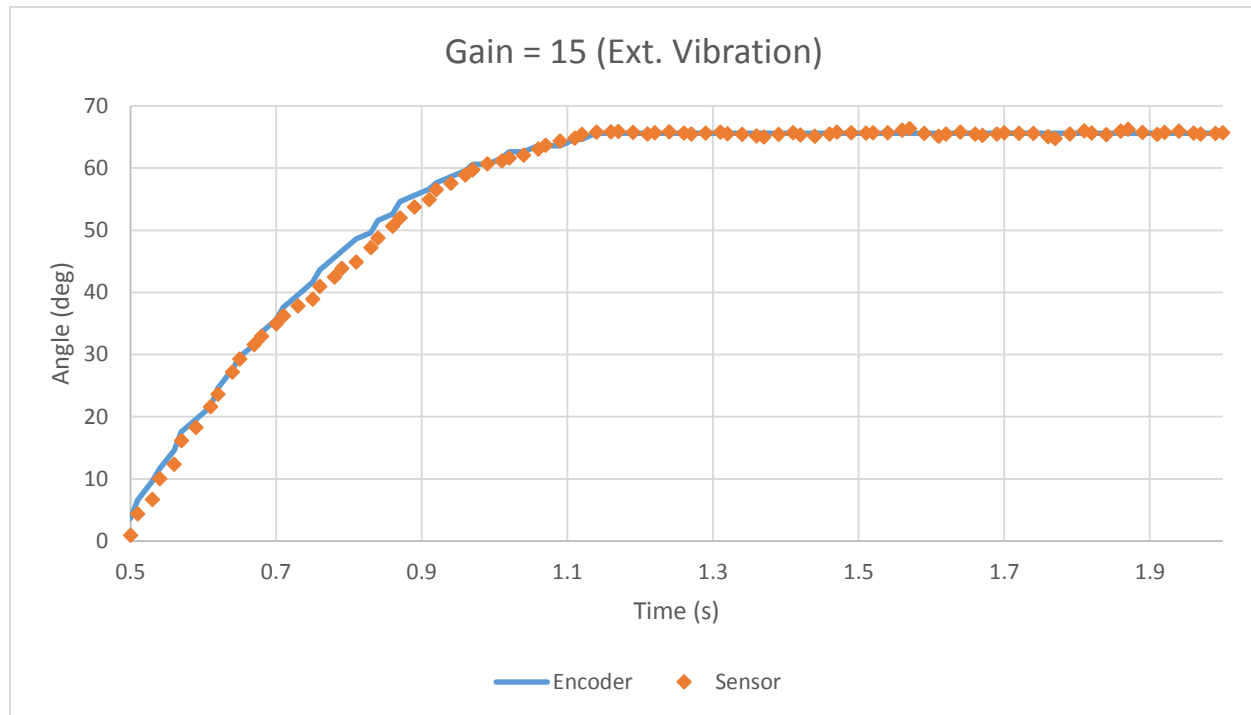


Figure 23: Dynamic Angle Testing: Gain=15 with External Vibrations

6.2 Motor Thrust Testing

As we developed the quadcopter flight controller code and considered PID implementation, concerns were raised that the motor thrust may not vary linearly with ESC command signal pulse length. If that were the case, the PID controllers would behave differently at different throttle settings unless some form of compensation was implemented, such as a motor speed mapping function.

To test the motor thrust and determine its relationship to ESC command signal, one of the motors was mounted to a load cell. The load cell had a 5 kg maximum load, more than adequate for the less than 1 kg maximum that we expected from the motor.

To measure the differential voltage across the load cell Wheatstone bridge and simultaneously digitize the reading, we used an [ADS1115 16-bit analog-to-digital converter from Adafruit](#). This ADC, seen in Figure 24, is I²C compatible, is capable of measuring either single-ended or differential voltages, and includes a built-in programmable gain amplifier (up to 16x).

After the load cell readings were calibrated using proof masses, an Arduino program was written that would increase the motor speed by steps of 100 μ s and take 100 measurements from the load cell at each speed. The measurements were then averaged to obtain a thrust reading at that speed setting. This test was run for two trials.

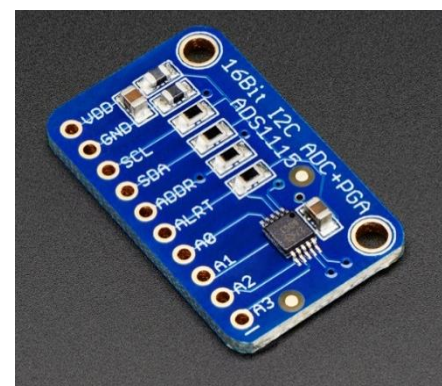


Figure 24: ADS1115 16-bit ADC from Adafruit

The results of this test were plotted in Excel and may be seen in Figure 25. While the thrust curve is not 100% linear, it is fairly linear in the middle portion of the curve. Because the quad was expected to hover at half-throttle, we were satisfied with the linearity enough to feel confident that the non-linearity would not affect the quad's stability.

The test also showed that the motor put out approximately 0.625 kg of thrust at maximum speed. This is very close to the motor's rating of 0.632 kg with the propellers that we chose.

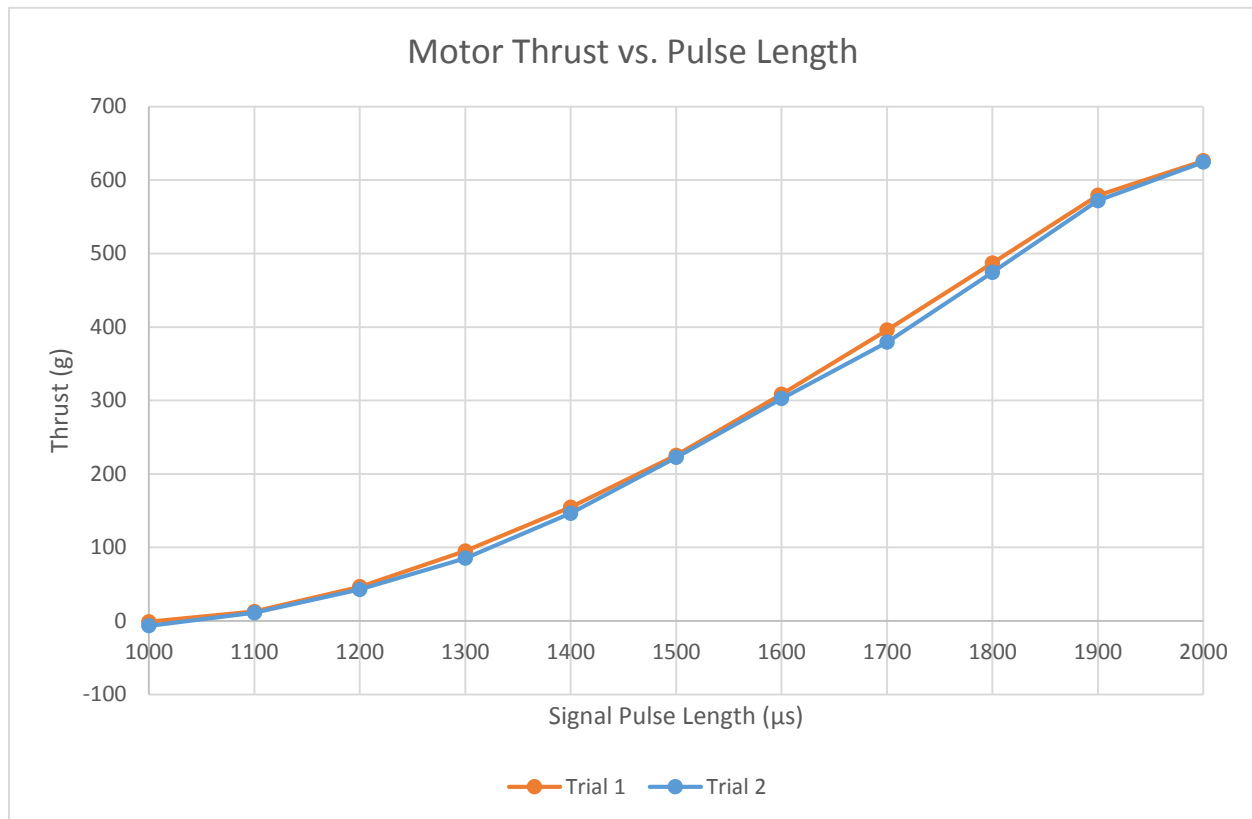


Figure 25: Motor Thrust Test Results

6.3 Stabilization Testing & PID Tuning

Once the quadcopter had been assembled, we needed to test its stabilization capabilities. To do this safely, we constructed a test stand that would restrain the quad while allowing it to rotate about one axis, either pitch or roll. The test stand, seen in Figure 26, also allowed us to operate the quad while keeping a USB cable connected for serial communications and fast program re-uploads.

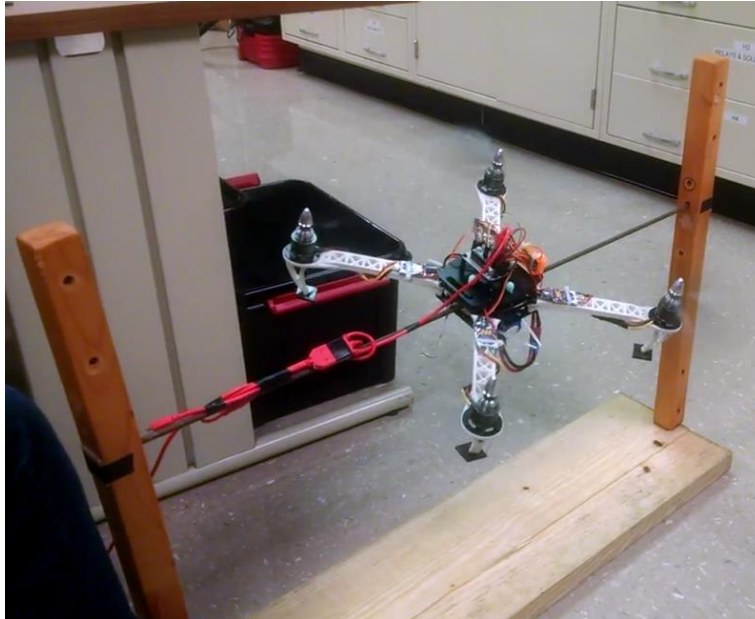


Figure 26: Quadcopter Single-Axis Test Stand

The quad was mounted to the pivot rod such that it was top-heavy, and therefore unstable. If the PID controllers did not perform as needed, the quad would flip over. This allowed us to test different PID coefficients for roll and pitch, and tune these parameters for the best performance.

Testing of various PID parameter configurations showed that with proportional control only, the quad was unable to maintain stability without oscillating. After adding derivative control, the quad became significantly more stable. A reasonable derivative also made the quad more responsive to inputs, as sharp changes in the RC input would produce spikes in derivative in the error and cause the quad to respond faster.

After significant testing, we determined that the following values of proportional and derivative control produced the best stabilization performance on the test stand:

$$K_P = 2.0 \quad K_D = 0.6$$

After determining these parameters, we experimented with adding integral control. While small values of K_I did not hurt the stabilization performance, higher values produced slow oscillations. In addition, including integral control adds the possibility of “wind-up,” which is large accumulation of integral error in one direction. While we managed to include some anti-wind-up measures in our code, we decided that the benefits of integral control were not significant enough for us to include it.

Video of quad stability testing on single-axis stand: <http://youtu.be/bEhCk-En4Bc>

6.4 In-Flight PID Tuning

After tuning the PID parameters on the test stand, we began to test the quadcopter in free flight. These tests were performed outdoors in a large open area, while maintaining safe distance from the quadcopter. We were confident after the single-axis tests that the quadcopter would be fairly stable in the roll and pitch axes, but we had been unable to test the yaw axis before free flight.

To begin, we attempted flight without any PID control on the yaw axis. While the quadcopter was able to fly thanks to the pitch and roll PIDs, it drifted significantly in the yaw direction. After enabling PID control on the yaw axis with proportional control only, the quad drifted much less in yaw and became much more controllable. We increased the proportional control term in the yaw PID until the quad was fairly responsive to yaw control inputs.

We found that the quad made small, fast oscillations in the roll and pitch axes in flight. To fix this, we lowered both the proportional and derivative terms on these axes. After tuning, we found that the following values yielded stable performance:

Roll/Pitch:	$K_P = 1.5$	$K_D = 0.3$	$K_I = 0$
Yaw:	$K_P = 3$	$K_D = 0$	$K_I = 0$



Figure 27: Still Frame from First Quadcopter Flight Video

Video of stable flight with PID values determined from test stand: <http://youtu.be/2CNzgS6Gww8>

Project Code

Project code is available as a compressed folder at
https://www.dropbox.com/s/chxlf21da600tjc/ME445_Quadcopter.zip?dl=0.

For compilation, the PinChangeInt library must be installed. PinChangeInt may be downloaded at
<https://code.google.com/p/arduino-pinchangeint/>.

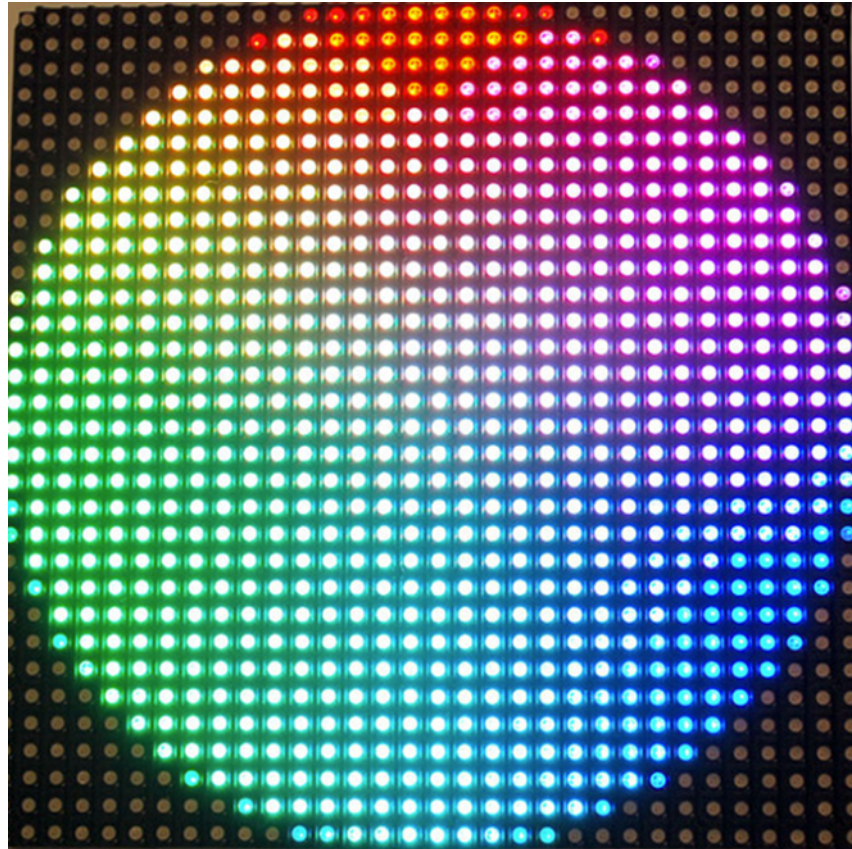
Bill of Materials

Purchased for Project			
Item	Qty	Indiv.	Sum
HobbyKing SK450 Frame	1	\$17.99	\$17.99
NTM 1000Kv Motor (short shaft)	4	\$15.99	\$63.96
NTM Prop Adapter	7	\$1.85	\$12.95
Afro ESC 20A	4	\$12.97	\$51.88
9x4.7 Carbon Fiber Props (1 pair)	3	\$6.80	\$20.40
XT60 to 4x3.5mm Bullet Breakout Cable	1	\$3.36	\$3.36
Arduino Pro Mini	1	\$9.95	\$9.95
DuBro Prop Balancing Stand	1	\$27.79	\$27.79
Sparkfun 9DOF Sensor Stick	1	\$49.95	\$49.95
		Total	\$258.23

Already Owned	
Item	Qty
Spektrum DX6i 2.4GHz Transmitter	1
OrangeRx 6Ch 2.4GHz Receiver	1
Turnigy 2200mAh 20C 3S LiPo	2
Turnigy Accucel-6 50W Charger	1

References

- A Guide To Using IMU (Accelerometer and Gyroscope Devices) in Embedded Applications. (n.d.). Retrieved 2014, from http://www.starlino.com/imu_guide.html
- How To Read an RC Receiver With A Microcontroller - Part 1. (n.d.). Retrieved from <http://rcarduino.blogspot.co.uk/2012/01/how-to-read-rc-receiver-with.html>
- Szafranski, G., & Czyba, R. (n.d.). Different Approaches of PID Control UAV Type Quadrotor. In Proceedings of the International Micro Air Vehicles Conference 2011 Summer Edition.



Audio Synchronized Light Show aka Blinky, the Blinding Light Panel

By: Michael Boyle and Ngan (Peter) Le

ME 445 Microcomputer Interfacing, Fall '14

Dr. Henry J. Sommer III

ABSTRACT

As time goes on, music is becoming more and more entrenched in computers and electronics. This begs the question of how audio signals can be represented by electronic signals. The complexity of a voice, let alone an entire band, will give rise to numerous overlapping harmonics. The challenge of audio processing then becomes separating these component parts to give a repeatable and predictable output. Understanding these processes reveals just how crucial frequency content analysis is to daily life.

In order to exemplify this concept, we integrated an electret microphone with variable resistance band pass filters. Using the outputs of different filters, we could output rhythmic light patterns created from any ambient noise, leading to a convenient product that requires no cable to output any music currently playing in the environment.

INTRODUCTION/MOTIVATION

When deciding on what project we wanted to undertake, the ultimate goal was to use music as an input. Looking at what the market currently had to offer, we found a particular Bluetooth speaker we wanted to emulate, the JBL Pulse (http://www.jbl.com/estore/jbl/us/products/JBL-Pulse/JBL%20PULSE_JBL_US). It is the first speaker we came across that had an integrated LED display. At first, the team was skeptical about how elegant a light show this portable speaker could display, but after seeing the Pulse in action, it was determined that the synchronized lights really added a completely new dimension to the already saturated market of Bluetooth speakers. Our goal was to figure out and use this new mix of entertainment.

This became the driving inspiration for the project. Having previous knowledge of the harmonic nature of physical sound, we decided that the panel could be controlled if we could separate harmonic components of a song's waveform. We have learned of the capabilities of filters to allow certain bands through and attenuate other signals. This theoretically should allow us to output rhythmic, predictable voltages corresponding to various harmonics passing through the specified filter. By implementing these filters, we hoped to better understand the frequency content of audio signals by being able to control which harmonics pass to the microcontroller. Once we do this, we will be able to control a 1024-pixel RGB LED display using a musical signal.

ACKNOWLEDGEMENTS

We want to take this opportunity to thank Dr. Sommer and Mike for all of their assistance. None of this would have been possible without Dr. Sommer's knowledge he has shared with us, as well as Mike's constant guidance and suggestions for improvement. He kept us on track and led us in the right direction.

ELECTRICAL SETUP

The electrical subsystems of this project are as follows: the Arduino Mega, the 32X32 RGB light panel, two variable resistance filters (one high pass, one low pass), and the electret microphone.

Due to the high number of analog inputs from the filters and the panel, the Arduino Mega was a necessary choice of microcontroller. Furthermore, this chip is much more conducive to the many different display modes that we will output. Each display requires a program that designs, addresses, and sets the LEDs based on defined ranges of electret output voltage. These programs get quite lengthy very quickly. For example, our mouth program has three images, yet required over a hundred lines of code to draw these specific shapes. Though our entire code was about 17 kilobytes, the Mega will have room (239 more kilobytes) to spare for any future displays. (Consult Appendix B for wiring of panel to controller).

The 32X32 Panel is the bulk of our electric components. To control the high number of LEDs with only a few pins, the panel employs 12 shift registers with 16 outputs each, for a total of 192 bits of information from one pin and a scan rate of 1:16. Furthermore, the panel is divided into 16 interleaved sections: each section being two rows (64 pixels/section * 3RGB = 192) . This allows the chip to drive 192 outputs from one pin. The Mega sets a 3-bit address for a section and it clocks out 192 and increments from address 0 to 7, at which point it sets the address back to #0. This means to change one light, the controller must redraw the whole panel, leading to no PWM capability and a process-intensive program. Additionally, this panel required a 2A, 5V power supply.

We needed to employ filters in order to separate the harmonics of the incoming waveform from the electret microphone. Because our program has set ranges of values between 0-1023 from the electret microphone, we needed to be able to adjust the cutoff frequencies of our high and low pass filter. For example, our mouth code should emphasize the main harmonic unique to the voice. By implementing a potentiometer in series with the filters, we can adjust the output voltage of the electret microphone to give values within the ranges desired. Example, if Billy Joel's voice exhibits repeated high frequency harmonics, we adjust our filter to be just below the cutoff frequency. This led to trial and error in fine-tuning the capacitance so as to not overshadow the variable resistance.

Lastly, our audio input was read by an electret microphone. This microphone sampled the entire analog waveform rather than simply being a pressure transducer and measuring volume. The range was 100Hz-10kHz, which covers most of the audible spectrum. It assigned a value from 0-1023 to the voltages coming in. Coupled with a high gain op-amp, this sensor was perfect for our needs of capturing all the overtones involved in a song.

PROGRAMMING

This project exclusively uses Arduino programming to perform the designated tasks by controlling the Arduino MEGA 2560. The reason an Arduino MEGA 2560 was ideal for this project was because it possesses many more analog and digital pins than the Arduino UNO has to offer. These extra pins were needed to allow for multiple switches to change between the modes, as well as using multiple inputs for the different analog filters that were implemented.

For this project to be possible, a couple libraries specific to this panel, which were obtained from the Adafruit website, had to be included in order to design special graphics. These libraries contain many capabilities, but only a few functions will be highlighted. These functions include:

- `matrix.fillRect(x,y,width,height,matrix.Color333(7,7,7));`
 - `fillRect` draws a rectangle and fills it with the desired color
 - `x` and `y` coordinates correspond to the top left corner of the rectangle
 - width and height of the desired rectangle
 - `matrix.Color333` controls the desired color by using RGB combinations
- `matrix.fillCircle(x,y,r,matrix.Color333(7,7,7));`
 - `fillCircle` draws a circle and fill it with desired color
 - `x` and `y` coordinates correspond to the center point of the circle
 - `r` corresponds to radius of the circle
- `matrix.drawPixel(x,y,matrix.Color333(7,7,7));`
 - `x` and `y` coordinates correspond to the desired pixel to be drawn
- `matrix.drawLine(x1,y1,x2,y2,matrix.Color333(7,7,7));`
 - `drawLine` draws a line using given starting and ending points
 - `x1` and `y1` coordinates correspond to the starting point of the line
 - `x2` and `y2` coordinates correspond to the ending point of the line

The panel contains 4 different modes that are programmed into the Arduino. 2 of the modes are continuous loops, and the other 2 require audio input from the microphone. These modes include:

- Colorwheel – this mode displays a color wheel that showcases the various colors that panel has to offer
- Plasma – this mode displays a randomized, ever changing, flow of colors by controlling the direction in which a colored pixel will travel
- Equalizer – this mode displays colored bars corresponding to frequency input
- Mouth – this mode displays a set of red lips that sync to a singing voice

(Consult Appendix C for complete code and comments)

DISCUSSION

Overall, the performance of the light panel achieved the goals we had set before we started this project. We wanted to have some sort of audio input to control the motion of our modes. The electret microphone did a very good job at outputting the voltages for our designs. Often times, too well and too fast. One of the biggest challenges we faced while working with this microphone was its sensitivity. This is where the filters come into play. With the filters, we were able to separate the signal into the desired bands needed for modes. However, working with these filters was no easy task. The filters were very time intensive because we had to manually adjust the filter using a potentiometer to receive the desired frequencies. Once we were able to adjust the filters correctly, we were able to refine the performance for our modes. For example, we were able to fine tune the high pass filter well enough to have one of our modes, called Mouth, to be able to sync to Billy Joel's voice from his song called "Miami 2017(I've Seen the Lights Go Out on Broadway)."

IMPROVEMENTS

Even though this project surpassed our expectations, there are still improvements that could be made. First and foremost, building a mount to attach the panel to would make it possible to showcase our panel in a more elegant manner. Another improvement would be to include additional filters and signals to achieve different frequencies for different graphics. Since the Arduino MEGA has a Flash Memory of 256KB, this leaves a significant amount of room for more modes to be implemented. We had a few other ideas for other modes, but were unable to make these ideas a reality.

-Talk about improvements if more time

-Making a mount

CONCLUSIONS

This product has a lot of potential as a graphic entertainment system. People spend up to thousands of dollars on these equalizers, and it is not hard to see why. The complex nature of the noise leads to high performance filter requirements if any repeatable output is to be measured.

Using filters we were successfully able to capture a particular range of harmonics. Our variable resistance potentiometers allow us to highlight what we want to hear, leading to a controllable entertainment system. This project takes

With greater resources and technology, we could have made an equalizer that has adjustable range and gain and fine tune the ranges to produce more fluid motion of the lights. The project exhibited what we wanted it to and we were able to take apart its components, but the complex nature of sound will always necessitate further refinement to truly take apart and understand these frequencies.

APPENDICES

Appendix A – Spec Sheets



Challenge Electronics

95 East Jefryn Boulevard
Deer Park, NY 11729

EMAIL: SALES@CHALLELEC.COM

Tel: 1-800-722-8197

1-631-595-2217

Fax: 1-631-586-5899

• ISO 9001:2000

• ISO 14001:2004

• ISO/TS 16949:2002

WEB: WWW.CHALLEGELECTRONICS.COM



PRODUCT INFORMATION

PART #	CEM-C9745JAD462P2.54R					Revision	0-2010
Omni-Directional Foil Electret Condenser Microphone							
DESCRIPTION							
Omni-Directional Foil Electret Microphone, 9.7 mm diameter and 4.5 mm high, Power Supply 5.0 V max, External Resistance Loading of 680 Ω, and sensitivity of -44 dB. Terminated with 2 solder points, Lead Free RoHS Compliant							
SPECIFICATIONS:							
Direction	Omni Directional Foil Electret				Minimum Direction sensitivity		
Operating Voltage Range	Vs= 1.0 Vdc ~ 10.0 Vdc				Power Supply (Vs)		1.5 V
Frequency Range	100 ~ 10,000 Hz.				Maximum Current		0.5 mA
Sensitivity	- 46 ± 2.0, (0 dB = 1V / Pa) at 1K Hz.				Minimum Sensitivity to Noise Ratio		58 dB
Sensitivity Reduction	3.0 V to 2.0 V -3 dB				Maximum input S.P.L.		110 dB at 1.0 KHz, THD <1%
Operating Temperature	-20°C to + 60°C				Storage Temperature		-40°C to + 75°C
Loading Resistance (R _L)	External, 680 Ω at Vs = 1.5 V, Max. 2,200 Ω				Built in Capacitors		None
Termination	PC Pins, 4.5 mm Long, 0.6 mm Ø, 2.54 mm Spacing						
Dimensions	Length / Diameter	9.7 mm Ø	Height	4.5 mm	Housing Material	Al-Mg Alloy.	Color
Approximate Weight	0.7 grams	Options				Compliance	RoHS, Lead Free
Reliability							
Thermal Operating Cycle Test	250 hours continuous operation at Rated Power, at Maximum Rated Operating Temperature *						
	250 hours continuous operation at Rated Power, at Minimum Rated Operating Temperature *						
Thermal Storage Cycle Test	Parts are subjected to 250 hours storage at Maximum Rated Storage Temperatures *						
	Parts are subjected to 250 hours storage at Minimum Rated Storage Temperatures *						
Thermal Shock Test:	Parts are subjected to five (5) cycles of Minimum and Maximum Operating Temperature. Each cycle shall be set per diagram below and is three (3) hours long *						
Humidity Test	Parts are subjected to 240 Hours at +40°C±2°C. 90-95% RH *						
Vibration Test	Parts are subjected to 2 Hours of at 1.5 mm with 10 to 55 Hz. vibration frequency to each of 3 perpendicular directions *						
Drop Test	Parts are dropped naturally from 1 meter height onto the surface of 40 mm wooden board, 2 axes (X,Y) directions, 3 times (6 times total) *						
Reliability Test Performance *	Parts should conform to original performance within ±5 dB tested with Rated Power, after 3 hours of recovery period.						
Termination Strength	Terminals should withstand a 1.0 Kg. pull test for up to 1 minute.						
Life Test	At rated voltage in room temperature continuously for 1,000 hours						
Warranty	For a period of one (1) year from date of shipping under normal operations conditions						
Typical Frequency Response			Microphone Response Toll Window		Dimensions Units in: mm Tolerance: ±0.3 mm		
			Frequency (Hz)	Lower Limit (dB)	Upper Limit (dB)		
			50	-6	+3		
			100	-3	+3		
			800	-3	+3		
			1000	0	0		
			1200	-3	+3		
			3000	-3	+8		
			5000	-3	+8		
			10000	-8	+8		

The information contained herein is believed to be correct, but no guarantee or warranty, express or implied, with respect to accuracy, completeness or results is extended and no liability is assumed. Challenge Electronics reserves the right to make changes in any specification, data or material contained herein.

Copyright © 2010 Challenge Electronics



Challenge Electronics

95 East Jefryn Boulevard

Deer Park, NY 11729

EMAIL: SALES@CHALLELEC.COM

Tel: 1-800-722-8197

1-631-595-2217

Fax: 1-631-586-5899

• ISO 9001:2000

• ISO 14001:2004

• ISO/TS 16949:2002

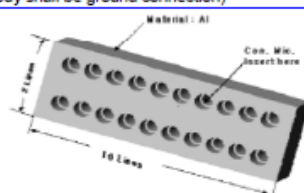


WEB: WWW.CHALLEGELECTRONICS.COM

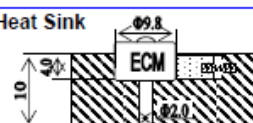
Soldering Instructions

1. Soldering temperature should be controlled under 320 and soldering time for each terminal should be 1~2 sec.
2. Microphone should be fixed on the metal block (heat sink), which has high radiation effects, and heat sink shall contact with MIC tightly.
3. Microphone may easily be destroyed by the static electricity. All countermeasure for eliminating static electricity must be executed (worktable and human body shall be ground connection)

Shape of heat sink



Shape of hole at fixed part Heat Sink



Packaging

1.1 Anti-Static Bag Parts



1.2 Small Box 100 Parts



1.3 Middle Box 6,000 Parts



1.4 Shipping Carton 12,000 Parts

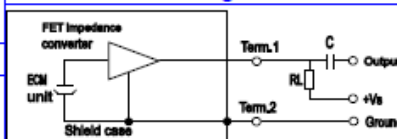


4. Shipping Label



1. Dimensions:	Length	Width	Height
1.1 Anti-Static Bag:	mm	mm	mm
1.2 Small Box:	100 mm	100 mm	5 mm
1.3 Middle Box:	450 mm	280 mm	135 mm
1.3 Carton Size:	550 mm	230 mm	235 mm
2. Quantity:	2.1 In Anti Static Box	100 parts	
	2.2 In mid. Size box	6,000 parts.	
	2.3 In master box	12,000 parts	
3. Weight:	3.1 One Part:	0.7 gram	
	3.2 Net Weight:	8.4 kg	
	3.3 Gross Weight:	12 kg	
4. Label Directions:	4.1 Contents should be visible clearly.		

Schematic Drawing



$R_L = 680 \Omega$

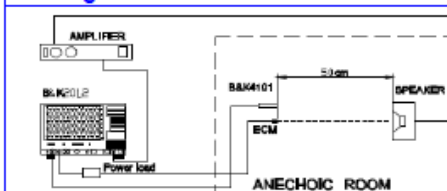
$V_S = 1.5 V$

$C = 1 \mu F$

Construction Materials

#	Name	Material	QTY
1	Dustproof gauze		1
2	Case	Al-Mg Alloy	1
3	Diaphragm	DUPONT	1
4	Spacer		1
5	Electret Plate	Copper blank	
6	Housing Chamber		1
7	PCB	FR4	1
8	FET		1
9	PC Pins		2

Testing Procedure



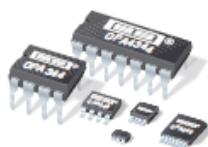
1. Measure the microphones under standard operating condition.
 2. Put the microphone and standard microphone face to the sound source (speaker), the distance between sound source and microphone & standard microphone is 50cm. And keep the center distance 5cm between them to ensure that the change of sound pressure should be kept within $\pm 1dB$.
 3. Keep the sound source pressure within $\pm 1dB$ from speaker Measured by standard microphone.
- The sensitivity of microphone can obtain its output voltage when sound source kept within 1,000Hz & 0.1Pa.

Testing Condition

In Normal Weather	In Arbitrate Weather
Environment Temperature: 5~+35°C	Environment Temperature: 20±2°C
Relative Humidity: 45 ~ 85%	Relative Humidity: 60 ~ 70%
Pressure: 86 ~ 106Kpa	Air Pressure: 86 ~ 106Kpa

The information contained herein is believed to be correct, but no guarantee or warranty, express or implied, with respect to accuracy, completeness or results is extended and no liability is assumed. Challenge Electronics reserves the right to make changes in any specification, data or material contained herein.

Copyright © 2010 Challenge Electronics



OPA344
OPA2344
OPA4344

OPA345
OPA2345
OPA4345

www.ti.com

SBOS107A – APRIL 2000 – REVISED AUGUST 2008

LOW POWER, SINGLE-SUPPLY, RAIL-TO-RAIL OPERATIONAL AMPLIFIERS

MicroAmplifier™ Series

FEATURES

- RAIL-TO-RAIL INPUT
- RAIL-TO-RAIL OUTPUT (within 1mV)
- LOW QUIESCENT CURRENT: 150µA typ
- MicroSIZE PACKAGES
 - SOT23-5
 - MSOP-8
 - TSSOP-14
- GAIN-BANDWIDTH
 - OPA344: 1MHz, $G \geq 1$
 - OPA345: 3MHz, $G \geq 5$
- SLEW RATE
 - OPA344: 0.8V/µs
 - OPA345: 2V/µs
- THD + NOISE: 0.006%

APPLICATIONS

- PCMCIA CARDS
- DATA ACQUISITION
- PROCESS CONTROL
- AUDIO PROCESSING
- COMMUNICATIONS
- ACTIVE FILTERS
- TEST EQUIPMENT

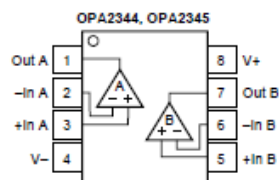
DESCRIPTION

The OPA344 and OPA345 series rail-to-rail CMOS operational amplifiers are designed for precision, low-power, miniature applications. The OPA344 is unity gain stable, while the OPA345 is optimized for gains greater than or equal to five, and has a gain-bandwidth product of 3MHz.

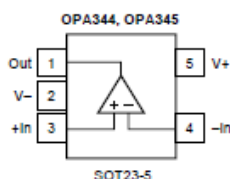
The OPA344 and OPA345 are optimized to operate on a single supply from 2.5V and up to 5.5V with an input common-mode voltage range that extends 300mV beyond the supplies. Quiescent current is only 250µA (max).

Rail-to-rail input and output make them ideal for driving sampling analog-to-digital converters. They are also well suited for general purpose and audio applications and providing I/V conversion at the output of D/A converters. Single, dual and quad versions have identical specs for design flexibility.

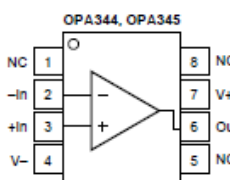
A variety of packages are available. All are specified for operation from -40°C to 85°C. A SPICE macromodel for design analysis is available for download from www.ti.com.



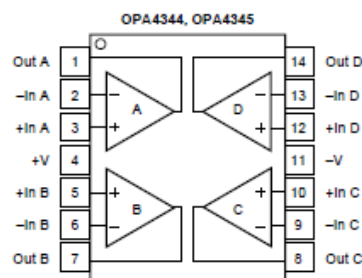
SO-8, MSOP-8, 8-Pin DIP (OPA2344 Only)



SOT23-5



SO-8, 8-Pin DIP (OPA344 Only)



TSSOP-14, SO-14, 14-Pin DIP (OPA4344 Only)



Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

All trademarks are the property of their respective owners.

PRODUCTION DATA Information is current as of publication date. Products conform to specifications per the terms of Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

SPECIFICATIONS: $V_S = 2.7V$ to $5.5V$

At $T_A = +25^\circ C$, $R_L = 10k\Omega$ connected to $V_S/2$ and $V_{OUT} = V_S/2$, unless otherwise noted.
 Boldface limits apply over the temperature range, $T_A = -40^\circ C$ to $+85^\circ C$.

PARAMETER	CONDITION	OPA344NA, UA, PA OPA2344EA, UA, PA OPA4344EA, UA, PA			UNITS
		MIN	TYP	MAX	
OFFSET VOLTAGE Input Offset Voltage Over Temperature vs Temperature vs Power Supply Over Temperature Channel Separation, dc $f = 1kHz$	V_{OS} $V_S = +5.5V$, $V_{CM} = V_S/2$ dV_{OS}/dT PSRR $V_S = 2.7V$ to $5.5V$, $V_{CM} < (V+) - 1.8V$ $V_S = 2.7V$ to $5.5V$, $V_{CM} < (V+) - 1.8V$		± 0.2 ± 0.8 ± 3 30 0.2 130	± 1 ± 1.2 200 250	mV mV $\mu V/^\circ C$ $\mu V/V$ $\mu V/V$ dB
INPUT BIAS CURRENT Input Bias Current Over Temperature Input Offset Current	I_B I_{OS}		± 0.2 See Typical Curve ± 0.2	± 10 ± 10	pA pA pA
NOISE Input Voltage Noise Input Voltage Noise Density Current Noise Density	e_n I_n $f = 0.1$ to $50kHz$ $f = 10kHz$ $f = 10kHz$		8 30 0.5		μV_{rms} nV/ \sqrt{Hz} fA/ \sqrt{Hz}
INPUT VOLTAGE RANGE Common-Mode Voltage Range Common-Mode Rejection Ratio Over Temperature Common-Mode Rejection Over Temperature Common-Mode Rejection Over Temperature	V_{CM} CMRR $V_S = +5.5V$, $-0.3V < V_{CM} < (V+) - 1.8V$ $V_S = +5.5V$, $-0.3V < V_{CM} < (V+) - 1.8V$ CMRR $V_S = +5.5V$, $-0.3V < V_{CM} < 5.8V$ $V_S = +5.5V$, $-0.3V < V_{CM} < 5.8V$ CMRR $V_S = +2.7V$, $-0.3V < V_{CM} < 3V$ $V_S = +2.7V$, $-0.3V < V_{CM} < 3V$	-0.3 76 74 70 68 66 64	92 84 80	$(V+) + 0.3$ 	V dB dB dB dB dB dB
INPUT IMPEDANCE Differential Common-Mode			$10^{13} \parallel 3$ $10^{13} \parallel 6$		$\Omega \parallel pF$ $\Omega \parallel pF$
OPEN-LOOP GAIN Open-Loop Voltage Gain Over Temperature Over Temperature	A_{OL} $R_L = 100k\Omega$, $10mV < V_O < (V+) - 10mV$ $R_L = 100k\Omega$, $10mV < V_O < (V+) - 10mV$ $R_L = 5k\Omega$, $400mV < V_O < (V+) - 400mV$ $R_L = 5k\Omega$, $400mV < V_O < (V+) - 400mV$	104 100 96 90	122 120		dB dB dB dB
FREQUENCY RESPONSE Gain-Bandwidth Product Slew Rate Settling Time, 0.1% 0.01% Overload Recovery Time Total Harmonic Distortion + Noise	GBW SR $V_S = 5.5V$, 2V Step $V_S = 5.5V$, 2V Step $V_{IN} = G \times V_S$ THD+N $V_S = 5.5V$, $V_O = 3V_{pp}$, $G = 1$, $f = 1kHz$		1 0.8 5 8 2.5 0.006		MHz V/ μs μs μs μs %
OUTPUT Voltage Output Swing from Rail ⁽¹⁾ Over Temperature Over Temperature Short-Circuit Current Capacitive Load Drive	I_{SC} C_{LOAD} $R_L = 100k\Omega$, $A_{OL} \geq 96dB$ $R_L = 100k\Omega$, $A_{OL} \geq 104dB$ $R_L = 100k\Omega$, $A_{OL} \geq 100dB$ $R_L = 5k\Omega$, $A_{OL} \geq 96dB$ $R_L = 5k\Omega$, $A_{OL} \geq 90dB$		1 3 40 ± 15	10 10 400 400	mV mV mV mV mA
POWER SUPPLY Specified Voltage Range Operating Voltage Range Quiescent Current (per amplifier) Over Temperature	V_S I_Q $V_S = 5.5V$, $I_Q = 0$	2.7	2.5 to 5.5 150	5.5 250 300	V V μA μA
TEMPERATURE RANGE Specified Range Operating Range Storage Range Thermal Resistance SOT23-5 Surface Mount MSOP-8 Surface Mount 8-Pin DIP SO-8 Surface Mount TSSOP-14 Surface Mount 14-Pin DIP SO-14 Surface Mount	θ_{JA}	-40 -55 -65		85 125 150	$^\circ C$ $^\circ C$ $^\circ C$ $^\circ C/W$ $^\circ C/W$ $^\circ C/W$ $^\circ C/W$ $^\circ C/W$ $^\circ C/W$ $^\circ C/W$ $^\circ C/W$

NOTE: (1) Output voltage swings are measured between the output and power-supply rails.

SPECIFICATIONS: $V_S = 2.7V$ to $5.5V$

At $T_A = +25^\circ C$, $R_L = 10k\Omega$ connected to $V_O/2$ and $V_{OUT} = V_O/2$, unless otherwise noted.
Boldface limits apply over the temperature range, $T_A = -40^\circ C$ to $+85^\circ C$.

PARAMETER	CONDITION	OPA345NA, UA OPA2345EA, UA OPA4345EA, UA			UNITS
		MIN	TYP	MAX	
OFFSET VOLTAGE Input Offset Voltage Over Temperature vs Temperature vs Power Supply Over Temperature Channel Separation, dc $f = 1kHz$	V_{OS} $V_S = +5.5V, V_{CM} = V_O/2$ dV_{OS}/dT $PSRR$ $V_S = 2.7V$ to $5.5V, V_{CM} < (V_+) - 1.8V$ $V_S = 2.7V$ to $5.5V, V_{CM} < (V_+) - 1.8V$		± 0.2 ± 0.8 ± 3 30 0.2 130	± 1 ± 1.2 200 250	mV mV $\mu V/^\circ C$ $\mu V/V$ $\mu V/V$ dB
INPUT BIAS CURRENT Input Bias Current Over Temperature Input Offset Current	I_B I_{OS}		± 0.2 See Typical Curve ± 0.2	± 10 ± 10	pA pA pA
NOISE Input Voltage Noise Input Voltage Noise Density Current Noise Density	e_n I_n $f = 0.1$ to $50kHz$ $f = 10kHz$ $f = 10kHz$		8 30 0.5		μV_{rms} nV/\sqrt{Hz} fA/\sqrt{Hz}
INPUT VOLTAGE RANGE Common-Mode Voltage Range Common-Mode Rejection Ratio Over Temperature Common-Mode Rejection Ratio Over Temperature Common-Mode Rejection Ratio Over Temperature	V_{CM} CMRR CMRR CMRR CMRR $V_S = +5.5V, -0.3V < V_{CM} < (V_+) - 1.8V$ $V_S = +5.5V, -0.3V < V_{CM} < (V_+) - 1.8V$ $V_S = +5.5V, -0.3V < V_{CM} < 5.8V$ $V_S = +5.5V, -0.3V < V_{CM} < 5.8V$ $V_S = +2.7V, -0.3V < V_{CM} < 3V$ $V_S = +2.7V, -0.3V < V_{CM} < 3V$	-0.3 76 74 70 68 66 64	92 84 80	$(V_+) + 0.3$ 	V dB dB dB dB dB dB
INPUT IMPEDANCE Differential Common-Mode			$10^{13} \parallel 3$ $10^{13} \parallel 6$		$\Omega \parallel pF$ $\Omega \parallel pF$
OPEN-LOOP GAIN Open-Loop Voltage Gain Over Temperature Over Temperature	A_{OL} $R_L = 100k\Omega, 10mV < V_O < (V_+) - 10mV$ $R_L = 100k\Omega, 10mV < V_O < (V_+) - 10mV$ $R_L = 5k\Omega, 400mV < V_O < (V_+) - 400mV$ $R_L = 5k\Omega, 400mV < V_O < (V_+) - 400mV$	104 100 96 90	122 120		dB dB dB dB
FREQUENCY RESPONSE Gain-Bandwidth Product Slew Rate Settling Time, 0.1% 0.01% Overload Recovery Time Total Harmonic Distortion + Noise	GBW SR $G = 5, 2V$ Output Step $G = 5, 2V$ Output Step $V_{IN} = G \cdot V_S$ THD+N $V_S = 5.5V, V_O = 2.5V_{pp}, G = 5, f = 1kHz$		3 2 1.5 1.6 2.5 0.006		MHz V/ μs μs μs μs %
OUTPUT Voltage Output Swing from Rail ⁽¹⁾ Over Temperature Over Temperature Short-Circuit Current Capacitive Load Drive	I_{SC} C_{LOAD} $R_L = 100k\Omega, A_{OL} \geq 96dB$ $R_L = 100k\Omega, A_{OL} \geq 104dB$ $R_L = 100k\Omega, A_{OL} \geq 100dB$ $R_L = 5k\Omega, A_{OL} \geq 96dB$ $R_L = 5k\Omega, A_{OL} \geq 90dB$		1 3 40 ± 15	10 10 400 400	mV mV mV mV mA
POWER SUPPLY Specified Voltage Range Operating Voltage Range Quiescent Current (per amplifier) Over Temperature	V_S I_Q $V_S = 5.5V, I_Q = 0$	2.7	2.5 to 5.5 150	5.5 250 300	V V μA μA
TEMPERATURE RANGE Specified Range Operating Range Storage Range Thermal Resistance SOT23-5 Surface Mount MSOP-8 Surface Mount SO-8 Surface Mount TSSOP-14 Surface Mount SO-14 Surface Mount	θ_{JA}	-40 -55 -65		85 125 150	$^\circ C$ $^\circ C$ $^\circ C$ $^\circ C/W$ $^\circ C/W$ $^\circ C/W$ $^\circ C/W$ $^\circ C/W$

NOTE: (1) Output voltage swings are measured between the output and power-supply rails.

OPA344, 2344, 4344
 OPA345, 2345, 4345
 SBOS107A



Appendix B – Wiring

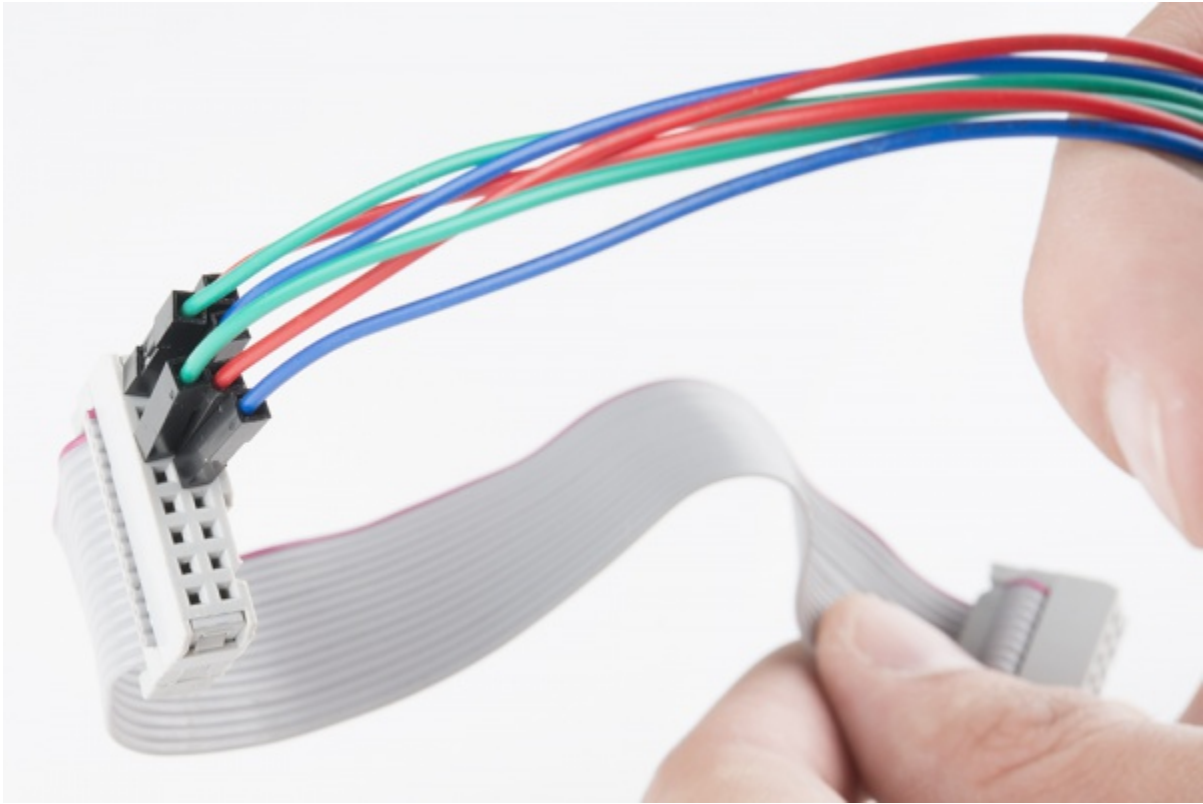


Figure 1: Ribbon Cable

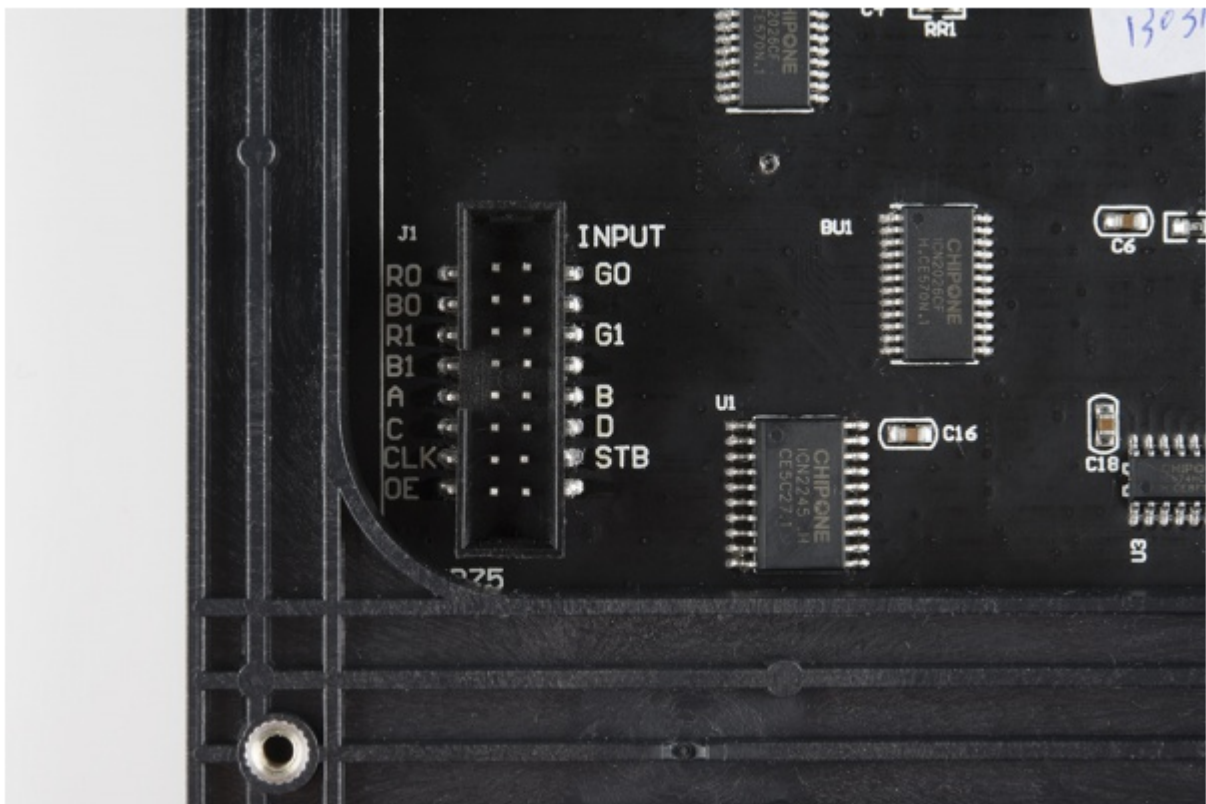


Figure 2: Panel Pins

Table 1: Panel Pins to Arduino Pins

Panel Pin Label	Panel Connector Pin #	Arduino Pin	Notes
R0	1	2	Red data (columns 1-16)
G0	2	3	Green data (columns 1-16)
B0	3	4	Blue data (columns 1-16)
GND	4	GND	Ground
R1	5	5	Red data (columns 17-32)
G1	6	6	Green data (columns 17-32)
B1	7	7	Blue data (columns 17-32)
GND	8	GND	Ground
A	9	A0	Demux input A0
B	10	A1	Demux input A1
C	11	A2	Demux input A2
D	12	A3	Demux input E1, E3 (32x32 panels only)
CLK	13	11	LED drivers' clock
STB	14	10	LED drivers' latch
OE	15	9	LED drivers' output enable
GND	16	GND	Ground

Using ribbon cable and some wires, it should be pretty easy to wire the Arduino and the panel together. Table 1 provides the corresponding Arduino pin for each of the panel pins.

Appendix C – Code

The code below combines all the functions that controlled the light panel into one single sketch. This code was uploaded onto the Arduino MEGA the day of the demonstrations.

```
// These libraries are specific to the Panel, and need to be included in every sketch
#include <Adafruit_GFX.h> // Core graphics library
#include <RGBmatrixPanel.h> // Hardware-specific library
#include <avr/pgmspace.h>

// If your 32x32 matrix has the SINGLE HEADER input,
// use this pinout:
#define CLK 11 // MUST be on PORTB!
#define OE 9
#define LAT 10
#define A A0
#define B A1
#define C A2
#define D A3
RGBmatrixPanel matrix(A, B, C, D, CLK, LAT, OE, false);

//Declare variables for Plasma
int Pswitch;
const float radius1 = 16.3, radius2 = 23.0, radius3 = 40.8, radius4 = 44.2,
        centerx1 = 16.1, centerx2 = 11.6, centerx3 = 23.4, centerx4 = 4.1,
        centery1 = 8.7, centery2 = 6.5, centery3 = 14.0, centery4 = -2.9;
float angle1 = 0.0, angle2 = 0.0, angle3 = 0.0, angle4 = 0.0;
long hueShift = 0;

//Declare variables for color changing for Plasma
static const int8_t PROGMEM sinetab[256] = {
    0, 2, 5, 8, 11, 15, 18, 21,
    24, 27, 30, 33, 36, 39, 42, 45,
    48, 51, 54, 56, 59, 62, 65, 67,
    70, 72, 75, 77, 80, 82, 85, 87,
    89, 91, 93, 96, 98, 100, 101, 103,
    105, 107, 108, 110, 111, 113, 114, 116,
    117, 118, 119, 120, 121, 122, 123, 123,
    124, 125, 125, 126, 126, 126, 126, 126,
```

```

127, 126, 126, 126, 126, 126, 125, 125,
124, 123, 123, 122, 121, 120, 119, 118,
117, 116, 114, 113, 111, 110, 108, 107,
105, 103, 101, 100, 98, 96, 93, 91,
89, 87, 85, 82, 80, 77, 75, 72,
70, 67, 65, 62, 59, 56, 54, 51,
48, 45, 42, 39, 36, 33, 30, 27,
24, 21, 18, 15, 11, 8, 5, 2,
0, -3, -6, -9, -12, -16, -19, -22,
-25, -28, -31, -34, -37, -40, -43, -46,
-49, -52, -55, -57, -60, -63, -66, -68,
-71, -73, -76, -78, -81, -83, -86, -88,
-90, -92, -94, -97, -99, -101, -102, -104,
-106, -108, -109, -111, -112, -114, -115, -117,
-118, -119, -120, -121, -122, -123, -124, -124,
-125, -126, -126, -127, -127, -127, -127, -127,
-128, -127, -127, -127, -127, -127, -126, -126,
-125, -124, -124, -123, -122, -121, -120, -119,
-118, -117, -115, -114, -112, -111, -109, -108,
-106, -104, -102, -101, -99, -97, -94, -92,
-90, -88, -86, -83, -81, -78, -76, -73,
-71, -68, -66, -63, -60, -57, -55, -52,
-49, -46, -43, -40, -37, -34, -31, -28,
-25, -22, -19, -16, -12, -9, -6, -3
};

//Declare variables for Equalizer
int Eswitch;
int Epin = 5;
int Echeck;
int Eincrement = 100;
int width = 4;

//Declare variables for Colorwheel
int Cswitch;

//Declare variables for Mouth
int Mswitch; //this will be the switch that will turn Mouth on

```

```
int Mpin = 5; //this will be the signal from the microphone for Mouth
int Mcheck; //this will read the analog signal from the microphone
int Mincrement = 50; //this is the increment in which Mouth is divided
int Delay = 10;
```

```
void setup(){
  matrix.begin();
  pinMode(2,INPUT); //digital pin for Plasma
  pinMode(3,INPUT); //digital pin for Equalizer
  pinMode(4,INPUT); //digital pin for Mouth
  pinMode(5,INPUT); //digital pin for Circle
}
```

```
void loop(){
  Pswitch = digitalRead(2);
  Eswitch = digitalRead(3);
  Mswitch = digitalRead(4);
  Cswitch = digitalRead(5);

  if(Pswitch == 1 & Eswitch==0 & Cswitch == 0 & Mswitch == 0){
    //if the switch for Plasma is engaged, run Plasma
    plasma();
  }
  else if(Cswitch == 1 & Eswitch==0 & Pswitch==0 & Mswitch == 0){
    //if the switch for Colorwheel is engaged, run Colorwheel
    colorwheel();
  }
  else if(Eswitch == 1 & Cswitch==0 & Pswitch == 0 & Mswitch == 0){
    //if the switch for Equalizer is engaged, run Equalizer
    Echeck = analogRead(Epin);
    equalizer();
  }
  else if(Mswitch == 1 & Cswitch==0 & Pswitch ==0 & Eswitch == 0){
    //if the switch for Mouth is engaged, run Mouth
    Mcheck = analogRead(Mpin);
    mouth();
  }
  else if(Mswitch == 0 & Cswitch==0 & Pswitch ==0 & Eswitch == 0){
```



```

//if all switches are 0, clear the screen
matrix.fillRect(0,0,32,32,matrix.Color333(0,0,0));
}
}

////////////////////////////////////
//These functions govern the motion of the modes above//
////////////////////////////////////
void plasma(){
    int      x1, x2, x3, x4, y1, y2, y3, y4, sx1, sx2, sx3, sx4;
    unsigned char x, y;
    long      value;

    sx1 = (int)(cos(angle1) * radius1 + centerx1);
    sx2 = (int)(cos(angle2) * radius2 + centerx2);
    sx3 = (int)(cos(angle3) * radius3 + centerx3);
    sx4 = (int)(cos(angle4) * radius4 + centerx4);
    y1 = (int)(sin(angle1) * radius1 + centery1);
    y2 = (int)(sin(angle2) * radius2 + centery2);
    y3 = (int)(sin(angle3) * radius3 + centery3);
    y4 = (int)(sin(angle4) * radius4 + centery4);

    for(y=0; y<matrix.height(); y++) {
        x1 = sx1; x2 = sx2; x3 = sx3; x4 = sx4;
        for(x=0; x<matrix.width(); x++) {
            value = hueShift
                + (int8_t)pgm_read_byte(sinetab + (uint8_t)((x1 * x1 + y1 * y1) >> 2))
                + (int8_t)pgm_read_byte(sinetab + (uint8_t)((x2 * x2 + y2 * y2) >> 2))
                + (int8_t)pgm_read_byte(sinetab + (uint8_t)((x3 * x3 + y3 * y3) >> 3))
                + (int8_t)pgm_read_byte(sinetab + (uint8_t)((x4 * x4 + y4 * y4) >> 3));
            matrix.drawPixel(x, y, matrix.ColorHSV(value * 3, 255, 255, true));
            x1--; x2--; x3--; x4--;
        }
        y1--; y2--; y3--; y4--;
    }

    angle1 += 0.03;
    angle2 -= 0.07;

```

```

angle3 += 0.13;
angle4 -= 0.15;
hueShift += 2;
}

void mouth(){
//default shape
if(Mcheck < (4*Mincrement)){
/////////Clear previous
//clear upper lip (from intermediate shape)
matrix.drawLine(5,16,26,16,matrix.Color333(0,0,0));
matrix.drawLine(7,15,24,15,matrix.Color333(0,0,0));
matrix.drawLine(10,14,14,14,matrix.Color333(0,0,0));
matrix.drawLine(17,14,21,14,matrix.Color333(0,0,0));
//clear lower lip (from intermediate shape)
matrix.drawLine(5,18,26,18,matrix.Color333(0,0,0));
matrix.drawLine(7,19,24,19,matrix.Color333(0,0,0));
matrix.drawLine(8,20,23,20,matrix.Color333(0,0,0));
matrix.drawLine(11,21,20,21,matrix.Color333(0,0,0));
/////clear upper lip (from open shape)
matrix.drawLine(6,16,7,16,matrix.Color333(0,0,0));
matrix.drawLine(8,16,23,16,matrix.Color333(0,0,0));
matrix.drawLine(24,16,25,16,matrix.Color333(0,0,0));
//second line
matrix.drawLine(7,15,9,15,matrix.Color333(0,0,0));
matrix.drawLine(10,15,21,15,matrix.Color333(0,0,0));
matrix.drawLine(22,15,24,15,matrix.Color333(0,0,0));
//third line
matrix.drawLine(8,14,23,14,matrix.Color333(0,0,0));
//fourth line
matrix.drawLine(10,13,14,13,matrix.Color333(0,0,0));
matrix.drawLine(17,13,21,13,matrix.Color333(0,0,0));
//between lips
matrix.drawPixel(6,17,matrix.Color333(0,0,0));
matrix.drawPixel(25,17,matrix.Color333(0,0,0));
//clear lower lip (from open shape)
//first line
matrix.drawLine(6,18,7,18,matrix.Color333(0,0,0));

```

```

matrix.drawLine(24,18,25,18,matrix.Color333(0,0,0));
//second line
matrix.drawLine(7,19,8,19,matrix.Color333(0,0,0));
matrix.drawLine(23,19,24,19,matrix.Color333(0,0,0));
//third line
matrix.drawLine(8,20,10,20,matrix.Color333(0,0,0));
// matrix.drawLine(10,20,21,20,matrix.Color333(5,0,0));
matrix.drawLine(21,20,23,20,matrix.Color333(0,0,0));
//fourth line
matrix.drawLine(9,21,22,21,matrix.Color333(0,0,0));
//fifth line
matrix.drawLine(11,22,20,22,matrix.Color333(0,0,0));
/////Set default (closed lips) shape
//upper lip
matrix.drawLine(5,17,26,17,matrix.Color333(5,0,0));
matrix.drawLine(7,16,24,16,matrix.Color333(5,0,0));
matrix.drawLine(10,15,14,15,matrix.Color333(5,0,0));
matrix.drawLine(17,15,21,15,matrix.Color333(5,0,0));
//lower lip
matrix.drawLine(5,17,26,17,matrix.Color333(5,0,0));
matrix.drawLine(7,18,24,18,matrix.Color333(5,0,0));
matrix.drawLine(8,19,23,19,matrix.Color333(5,0,0));
matrix.drawLine(11,20,20,20,matrix.Color333(5,0,0));
}

//start "mouth" opening
else if(Mcheck >= (4*Mincrement) & Mcheck < (5*Mincrement-1)){
    /////Clear previous (from default and open shape)
    /////clear upper lip (from default shape)
    matrix.drawLine(5,17,26,17,matrix.Color333(0,0,0));
    matrix.drawLine(7,16,24,16,matrix.Color333(0,0,0));
    matrix.drawLine(10,15,14,15,matrix.Color333(0,0,0));
    matrix.drawLine(16,15,20,15,matrix.Color333(0,0,0));
    /////clear lower lip (from default shape)
    matrix.drawLine(5,17,26,17,matrix.Color333(0,0,0));
    matrix.drawLine(7,18,24,18,matrix.Color333(0,0,0));
    matrix.drawLine(8,19,23,19,matrix.Color333(0,0,0));
    matrix.drawLine(11,20,20,20,matrix.Color333(0,0,0));
}

```

```

/////clear upper lip (from open shape)
matrix.drawLine(6,16,7,16,matrix.Color333(0,0,0));
matrix.drawLine(8,16,23,16,matrix.Color333(0,0,0));
matrix.drawLine(24,16,25,16,matrix.Color333(0,0,0));
//second line
matrix.drawLine(7,15,9,15,matrix.Color333(0,0,0));
matrix.drawLine(10,15,21,15,matrix.Color333(0,0,0));
matrix.drawLine(22,15,24,15,matrix.Color333(0,0,0));
//third line
matrix.drawLine(8,14,23,14,matrix.Color333(0,0,0));
//fourth line
matrix.drawLine(10,13,14,13,matrix.Color333(0,0,0));
matrix.drawLine(17,13,21,13,matrix.Color333(0,0,0));
//between lips
matrix.drawPixel(6,17,matrix.Color333(0,0,0));
matrix.drawPixel(25,17,matrix.Color333(0,0,0));
//clear lower lip (from open shape)
//first line
matrix.drawLine(6,18,7,18,matrix.Color333(0,0,0));
matrix.drawLine(24,18,25,18,matrix.Color333(0,0,0));
//second line
matrix.drawLine(7,19,8,19,matrix.Color333(0,0,0));
matrix.drawLine(23,19,24,19,matrix.Color333(0,0,0));
//third line
matrix.drawLine(8,20,10,20,matrix.Color333(0,0,0));
// matrix.drawLine(10,20,21,20,matrix.Color333(5,0,0));
matrix.drawLine(21,20,23,20,matrix.Color333(0,0,0));
//fourth line
matrix.drawLine(9,21,22,21,matrix.Color333(0,0,0));
//fifth line
matrix.drawLine(11,22,20,22,matrix.Color333(0,0,0));
////////Begin opening mouth
//upper lip
matrix.drawLine(5,16,26,16,matrix.Color333(5,0,0));
matrix.drawLine(7,15,24,15,matrix.Color333(5,0,0));
matrix.drawLine(10,14,14,14,matrix.Color333(5,0,0));
matrix.drawLine(17,14,21,14,matrix.Color333(5,0,0));
//gap between lips

```

```

matrix.drawLine(5,17,26,17,matrix.Color333(0,0,0));
//lower lip
matrix.drawLine(5,18,26,18,matrix.Color333(5,0,0));
matrix.drawLine(7,19,24,19,matrix.Color333(5,0,0));
matrix.drawLine(8,20,23,20,matrix.Color333(5,0,0));
matrix.drawLine(11,21,20,21,matrix.Color333(5,0,0));
}

else if(Mcheck >= 5*Mincrement & Mcheck < (6*Mincrement-1)){
/////Clear previous shape
//clear upper lip (intermediate shape)
matrix.drawLine(5,16,26,16,matrix.Color333(0,0,0));
matrix.drawLine(7,15,24,15,matrix.Color333(0,0,0));
matrix.drawLine(10,14,14,14,matrix.Color333(0,0,0));
matrix.drawLine(16,14,20,14,matrix.Color333(0,0,0));
//clear lower lip (intermediate shape)
matrix.drawLine(5,18,26,18,matrix.Color333(0,0,0));
matrix.drawLine(7,19,24,19,matrix.Color333(0,0,0));
matrix.drawLine(8,20,23,20,matrix.Color333(0,0,0));
matrix.drawLine(11,21,20,21,matrix.Color333(0,0,0));
/////clear upper lip (from default shape)
matrix.drawLine(5,17,26,17,matrix.Color333(0,0,0));
matrix.drawLine(7,16,24,16,matrix.Color333(0,0,0));
matrix.drawLine(10,15,14,15,matrix.Color333(0,0,0));
matrix.drawLine(16,15,20,15,matrix.Color333(0,0,0));
/////clear lower lip (from default shape)
matrix.drawLine(5,17,26,17,matrix.Color333(0,0,0));
matrix.drawLine(7,18,24,18,matrix.Color333(0,0,0));
matrix.drawLine(8,19,23,19,matrix.Color333(0,0,0));
matrix.drawLine(11,20,20,20,matrix.Color333(0,0,0));
//Begin open mouth shape
////////////////////
//upper lip
//firstline
matrix.drawLine(6,16,7,16,matrix.Color333(5,0,0));
matrix.drawLine(8,16,23,16,matrix.Color333(7,7,7));
matrix.drawLine(24,16,25,16,matrix.Color333(5,0,0));
//second line

```

```

matrix.drawLine(7,15,9,15,matrix.Color333(5,0,0));
matrix.drawLine(10,15,21,15,matrix.Color333(7,7,7));
matrix.drawLine(22,15,24,15,matrix.Color333(5,0,0));
//third line
matrix.drawLine(8,14,23,14,matrix.Color333(5,0,0));
//fourth line
matrix.drawLine(10,13,14,13,matrix.Color333(5,0,0));
matrix.drawLine(17,13,21,13,matrix.Color333(5,0,0));
//between lips
matrix.drawPixel(6,17,matrix.Color333(5,0,0));
matrix.drawPixel(25,17,matrix.Color333(5,0,0));
////
//lower lip
//first line
matrix.drawLine(6,18,7,18,matrix.Color333(5,0,0));
matrix.drawLine(24,18,25,18,matrix.Color333(5,0,0));
//second line
matrix.drawLine(7,19,8,19,matrix.Color333(5,0,0));
matrix.drawLine(23,19,24,19,matrix.Color333(5,0,0));
//third line
matrix.drawLine(8,20,10,20,matrix.Color333(5,0,0));
// matrix.drawLine(10,20,21,20,matrix.Color333(5,0,0));
matrix.drawLine(21,20,23,20,matrix.Color333(5,0,0));
//fourth line
matrix.drawLine(9,21,22,21,matrix.Color333(5,0,0));
//fifth line
matrix.drawLine(11,22,20,22,matrix.Color333(5,0,0));
}
}

void colorwheel(){
    int    a, b, hue;
    float  da, db, d;
    uint8_t sat, val;
    uint16_t c;

    for(b=0; b < matrix.width(); b++) {
        db = 15.5 - (float)b;

```

```

for(a=0; a < matrix.height(); a++) {
    da = 15.5 - (float)a;
    d = da * da + db * db;
    if(d <= (16.5 * 16.5)) { // Inside the circle(ish)?
        hue = (int)((atan2(-db, da) + PI) * 1536.0 / (PI * 2.0));
        d = sqrt(d);
        if(d > 15.5) {
            // Do a little pseudo anti-aliasing along perimeter
            sat = 255;
            val = (int)((1.0 - (d - 15.5)) * 255.0 + 0.5);
        } else
        {
            // White at center
            sat = (int)(d / 15.5 * 255.0 + 0.5);
            val = 255;
        }
        c = matrix.ColorHSV(hue, sat, val, true);
    } else {
        c = 0;
    }
    matrix.drawPixel(a, b, c);
}
}

void equalizer(){
    //clear all bars if less than 200
    if(Echeck < (2*Eincrement)){
        Cblue();
        Cred();
        Cgreen();
        Cyellow();
        Corange();
        Cmagenta();
        Caqua();
        Cfuscia();
        delay(Delay); //this delay is used to mitigate the flickering
    }
}

```

```

//start Echeck sequence
else if(Echeck >= (2*Eincrement) & Echeck < (3*Eincrement-1)){
    //clear red, green, yellow, orange, magenta, aqua, and fuscia is less than 300
    Cred();
    Cgreen();
    Cyellow();
    Corange();
    Cmagenta();
    Caqua();
    Cfuscia();
    //fill with blue bar
    blue();
    delay(Delay);
}

else if(Echeck >= 3*Eincrement & Echeck < (4*Eincrement-1)){
    //clear green, yellow, orange, magenta, aqua, and fuscia is less than 400
    Cgreen();
    Cyellow();
    Corange();
    Cmagenta();
    Caqua();
    Cfuscia();
    //fill with blue and red bars
    blue();
    red();
    delay(Delay);
}

else if(Echeck >= 4*Eincrement & Echeck < (5*Eincrement-1)){
    //clear yellow, orange, magenta, aqua, and fuscia is less than 500
    Cyellow();
    Corange();
    Cmagenta();
    Caqua();
    Cfuscia();
    //fill with blue, red, and green bars

```



```
blue();  
red();  
green();  
delay(Delay);  
}
```

```
else if(Echeck >= 5*Eincrement & Echeck < (6*Eincrement-1)){  
    //clear orange, magenta, aqua, and fuscia is less than 600  
    Corange();  
    Cmagenta();  
    Caqua();  
    Cfuscia();  
    //fill with blue, red, green, and yellow bars  
    blue();  
    red();  
    green();  
    yellow();  
    delay(Delay);  
}
```

```
else if(Echeck >= 6*Eincrement & Echeck < (7*Eincrement-1)){  
    //clear magenta, aqua, and fuscia is less than 700  
    Cmagenta();  
    Caqua();  
    Cfuscia();  
    //fill with blue, red, green, yellow, and orange bars  
    blue();  
    red();  
    green();  
    yellow();  
    orange();  
    delay(Delay);  
}
```

```
else if(Echeck >= 7*Eincrement & Echeck < (8*Eincrement-1)){  
    //clear aqua and fuscia is less than 800  
    Caqua();  
    Cfuscia();
```

```

//fill with blue, red, green, yellow, orange, and magenta bars
blue();
red();
green();
yellow();
orange();
magenta();
delay(Delay);
}

```

```

else if(Echeck >= 8*Eincrement & Echeck < (9*Eincrement-1)){
//clear fuschia if less than 900
Cfuschia();
//fill with blue, red, green, yellow, orange, magenta, and aqua bars
blue();
red();
green();
yellow();
orange();
magenta();
aqua();
delay(Delay);
}

```

```

else if(Echeck >= 9*Eincrement){
//fill with blue, red, green, yellow, orange, magenta, and aqua bars
blue();
red();
green();
yellow();
orange();
magenta();
aqua();
fuschia();
delay(Delay);
}

```

//This delay is used to slow the readings down if the panel flashes too much

```
//delay(75);  
}
```

```
//these functions will fill the bars
```

```
void blue(){matrix.fillRect(0, 28, width, 4, matrix.Color333(0, 0, 4));}  
void red(){matrix.fillRect(4, 24, width, 12, matrix.Color333(4, 0, 0));}  
void green(){matrix.fillRect(8, 20, width, 16, matrix.Color333(0, 4, 0));}  
void yellow(){matrix.fillRect(12, 16, width, 20, matrix.Color333(4, 4, 0));}  
void orange(){matrix.fillRect(16, 12, width, 24, matrix.Color333(4, 1, 0));}  
void magenta(){matrix.fillRect(20, 8, width, 28, matrix.Color333(4, 0, 4));}  
void aqua(){matrix.fillRect(24, 4, width, 32, matrix.Color333(0, 4, 4));}  
void fuscia(){matrix.fillRect(28, 0, width, 32, matrix.Color333(4, 0, 1));}
```

```
//these functions will clear the bars
```

```
void Cblue(){matrix.fillRect(0, 28, width, 4, matrix.Color333(0, 0, 0));}  
void Cred(){matrix.fillRect(4, 24, width, 12, matrix.Color333(0, 0, 0));}  
void Cgreen(){matrix.fillRect(8, 20, width, 16, matrix.Color333(0, 0, 0));}  
void Cyellow(){matrix.fillRect(12, 16, width, 20, matrix.Color333(0, 0, 0));}  
void Corange(){matrix.fillRect(16, 12, width, 24, matrix.Color333(0, 0, 0));}  
void Cmagenta(){matrix.fillRect(20, 8, width, 28, matrix.Color333(0, 0, 0));}  
void Caqua(){matrix.fillRect(24, 4, width, 32, matrix.Color333(0, 0, 0));}  
void Cfuscia(){matrix.fillRect(28, 0, width, 32, matrix.Color333(0, 0, 0));}
```

THE PENNSYLVANIA STATE UNIVERSITY

ME 445 Automatic Bicycle Transmission

An Exercise in Automatic Control Systems

Devin O'Donnell and Donald Bradfield

12/19/2014



[Type the abstract of the document here. The abstract is typically a short summary of the contents of the document. Type the abstract of the document here. The abstract is typically a short summary of the contents of the document.]

Table of Contents

1.0 Introduction.....	2
Design Objectives.....	2
2.0 Explanation of Design.....	2
2.1 System Design.....	2
<i>Relationship of Components</i>	3
2.2 Components.....	3
2.2.1 Arduino.....	3
2.2.2 Hall Effect Sensor.....	3
2.2.3 Accelerometer.....	4
2.2.4 Servo Motor.....	4
2.2.5 Derailleur and Spring Assembly.....	5
2.2.6 Power.....	6
2.3 Cost.....	7
3.0 Explanation of Arduino Code.....	8
3.1 Gear Shifting Algorithm.....	8
<i>Flowchart of Logic</i>	9
3.2 Explanation of Code for Incorporating Specific Components.....	10
3.2.1 Hall Effect Sensor.....	10
3.2.2. MPU-6050.....	10
3.3.3 Servo Motor.....	10
3.3. Component Adjustment and Synergy.....	10
3.3.1 Derailleur and Servo Position.....	10
3.3.2 Hall Effect Sensor and Voltage Drop.....	10
4.0 Final. Results.....	11
4.1 Lessons Learned.....	11
4.2 Conclusion.....	11
4.3 Arduino Code.....	12
4.4 Available Datasheets for Major Components.....	15
4.4.1 HS-805BB.....	15
4.4.2 MPU-6050.....	17

1.0 Introduction

The goal of our project was to design and build an automatic bicycle transmission that would allow a user to have a comfortable road riding experience without having to manually shift gears. We wanted to build a robust system that would be able to respond appropriately by shifting to whatever gear the user required. The basic concept is that as the user experiences more difficulty the user will require a lower gear, but if he has high momentum, he'll want a higher gear to maximize speed. One large factor that would affect his momentum is the incline of the hill he is on. Our basic design is to use the Arduino to incorporate bicycle crankshaft speed and incline to make a decision on how to move an actuator that will shift the gears.

Our Design Objectives

Our team sought out to design and fabricate an automatic bicycle transmission for a relative low cost. The bicycle was designed to provide the user with an optimal riding experience, by adjusting the gearing of the bicycle to suit the riders preferred pedaling speed (which was found experimentally to be 50 to 80 RPM). We also hoped to be able to incorporate a way to use the accelerometer to use the incline to adjust the shifting algorithm.

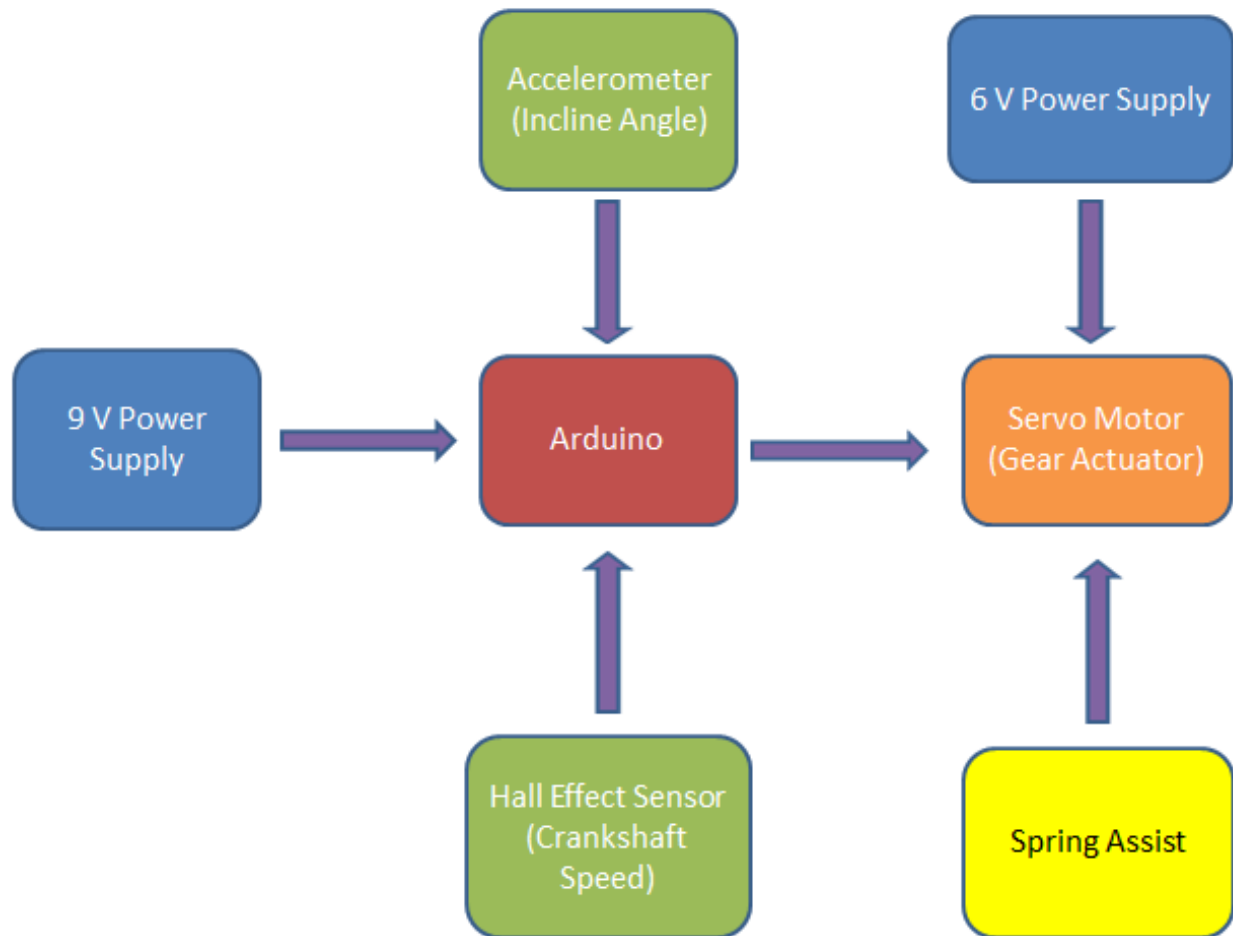
2.0 Explanation of Design

2.1 System Design

Below is the layout for all the major components for our system. As we stated above, the Arduino uses two inputs for the code that gives output commands to the servo motor. The servo motor is attached to the derailleur cable, and pulls on the cable to hold the derailleur in the appropriate gear position. Because the return spring originally installed in the derailleur is so strong, we have added an additional spring to pull the servo motor in the opposite direction. The servo requires a 6 V power supply, while the Arduino ideally uses 9 V. We have therefore used two separate battery packs to power them. For more details on individual components and the gear-shifting algorithm in our Arduino code, please read below.

Note: You will notice when you inspect our bicycle that our system only controls the fine-control gears that originally were placed on the right-hand side of the handle bars. For the purposes of our project, the left-hand gear shifter is to remain untouched.

Relationship of Components:



2.2 Components

2.2.1 Arduino:

We used a Sparkfun Redboard for our project because it was easier to interface to the wires with the breakout pins.

2.2.2 Hall Effect Sensor:

We used a radiometric sensor that had an analog output because it was readily available and easy to use in our code to tell when the magnet on the crankshaft is passed by it.



2.2.3 Accelerometer:

We used an MPU-6050 accelerometer-gyroscope combo. We decided to use only the accelerometer component because we were having difficulties programming the gyroscope and felt that the accuracy and response time of the accelerometer were more than sufficient for a riding going up and down slopes.



2.2.4 Servo Motor:

The HS-805BB servo motor was a good low-voltage (6 V) solution to moving the gear derailleur. The HS-805BB was selected because it was in the lab and thought to be powerful enough to overcome the return-spring in the derailleur. We found that it was not powerful enough on its own after we had ordered one for our project. We were able to develop a spring-assist assembly that simply added a spring that pulled in the opposite direction of the return spring. In the end this assembly made the lower-power motor and lowered our power-supply requirements.

2.2.5 Derailleur and Spring Assembly:

The assembly involved in shifting the gears included the derailleur, the cable that runs from the derailleur to the motor, and the spring that pulls the motor in the opposite direction of the derailleur spring.





2.2.6 Power:

We found that Sparkfun Redboard works best with a 9 V power supply and we used one of the Tenenergy rechargeable battery packs from the lab. The HS-805BB called for 6 V, and we were able to provide that with 4 AA battery holder and batteries from the instrumentation room in Reber.



2.3 Cost

Most of the electronic components were available in the lab. The motor had to be ordered online, and the springs and washers were bought from Home Depot. The bike itself was bought on Craigslist. Below is a BOM of the cost of our system based on estimate prices from online.

Bill of Materials				
Item	Source	Cost Per Unit (\$)	Quantity	Total
Sparkfun Redboard	sparkfun.com	19.95	1	19.95
4 Industrial AA Batteries	Leadmask.com	2.50	1	2.50
Tenergy 9V rechargeable	Ali-battery.com	4.19	1	
MPU-6050	MiniInTheBox.com	5.00	1	5.00
Radiometric Hall Effect Sensor	Verical.com	2.00	1	2.00
HS-805BB Servo Motor	Servocity.com	39.99	1	39.99
Spring Set	Home Depot	3.84	1	3.84
Hose Clamp	Home Depot	1.85	2	3.70
Shacles for Cables	Home Depot	1.68	2	3.36
4 AA Battery Holder	MiniInTheBox.com	1.99	1	1.99
9V Battery Holder	ebay.com	2.50	1	2.50
Turnbuckle	Home Depot	2.56	1	2.56
Total				87.39

3.0 Explanation of Arduino Code

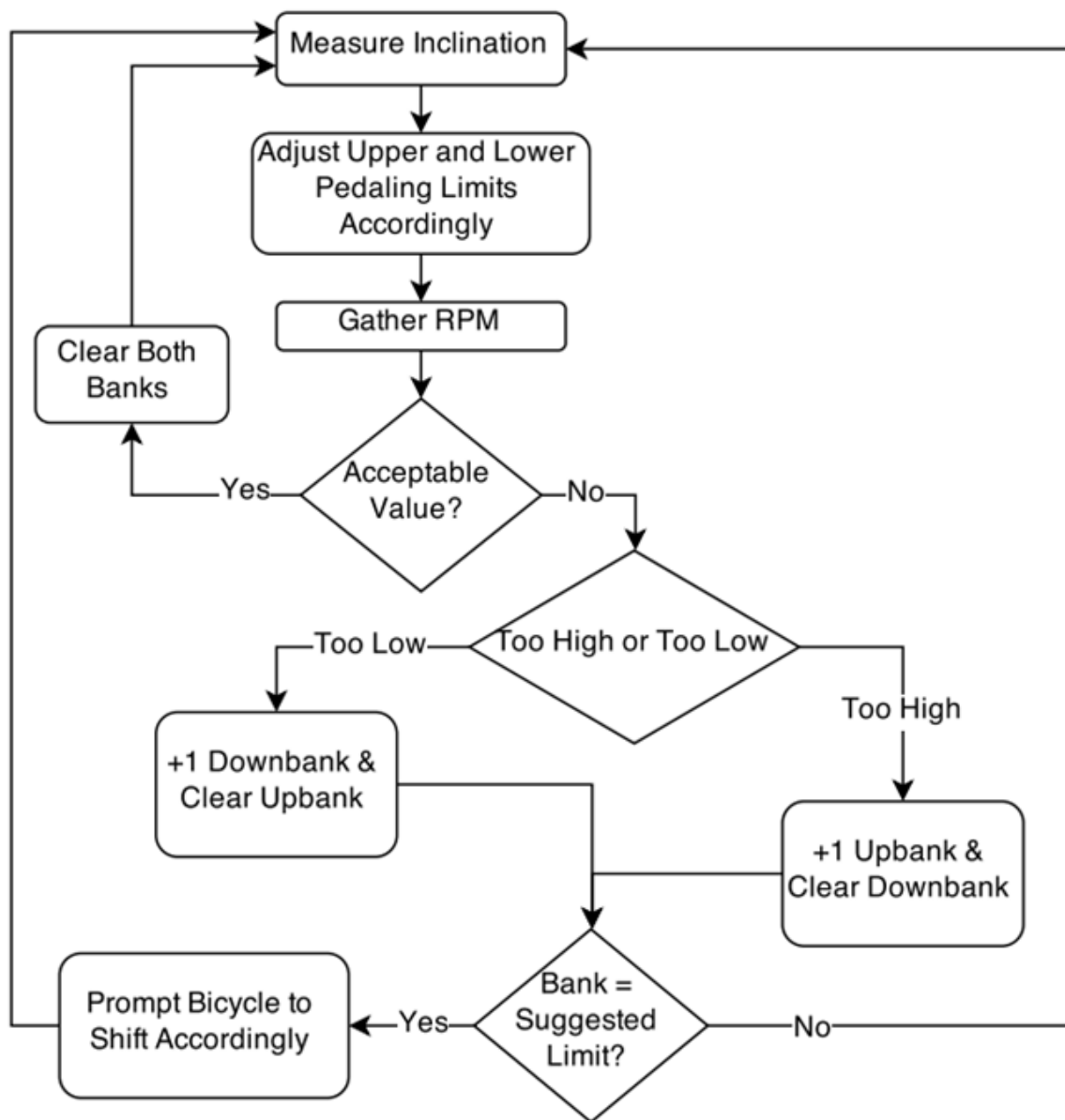
3.1 Gear Shifting Algorithm

The basic method we used to balance user difficulty level with speed was to monitor the crankshafts speed. Our main goal then is to keep the resistance to peddling as high as possible within a comfortable level. We assumed that if the resistance to peddling from lack of momentum is high, the user's biological feedback system will cause them to slow pedaling. This is measured by our Hall-effect sensor, and if the peddling drops below a certain lower bound rpm, indicating an uncomfortable resistance, Arduino decides to give the user more torque and tell the servo-motor to shift to a lower gear. If conversely, the rpm's measured by the Hall-effect sensor cross the upper bound, the Arduino basically interprets it as excess comfort, and orders the servo motor to shift up to increase speed.

The accelerometer is used to measure the incline that the bike is on, and the Arduino uses the incline angle to scale the lower and upper bounds in its code. Although an incline or a decline could cause the crankshaft speed to increase or decrease on its own, and therefore cause the appropriate shifting, we thought that the user would appreciate more leniency towards downshifting if they were going up a hill. If the lower bound is reduced when the incline increases, the bike has a better chance of shifting down sooner, before the user gets stopped by gravity mid-peddle or ends up shifting very hard. The incline is used to calculate a gain to multiply both the lower and upper bounds by. Since the downward force on an object on an incline increases at a function of $\sin(\theta)$, the sine function is used to calculate this gain.

Since our system uses the crankshaft speed to make decisions about shifting, we wanted to make sure the Arduino didn't interpret coasting (lack of peddling) as a need to downshift. We also didn't want the system to try to force its way into first gear while sitting at a stop sign. To solve this, if a turn of the crankshaft takes too long the Arduino does not record that turn as an rpm to use in its algorithm. While the user is coasting, no crankshaft speed measurements are recorded because the magnets is not passing by the Hall-effect sensor. When the user begins peddling again, that rpm recording is discounted. As the user continues to peddle, the Arduino will record more readings and operate as normal. In order to keep the system from being too responsive, it will not shift up or down without two "votes" to do so.

Flowchart of Logic:



3.2 Explanation of Code for Incorporating Specific Components

3.2.1 Hall-effect Sensor:

The Hall-effect sensor is used to calculate crankshaft speed in void loop(). Every the magnet passes by the hall effect sensor, the change in time since the last instance is calculated and used to determine rpm. This is then used in the shifting algorithm.

3.2.2 MPU-6050:

We created the function, SLOPE() to determine the incline angle with the MPU-6050 and determine the gain to multiply the bounds by. Although the MPU-6050 has a MEMS gyroscope, we had serious trouble incorporating it to find the angle. We were able to use the MEMS accelerometer fairly easily and found it had a sufficient response time for our application. We imported the Wire library to read raw X and Y acceleration components and imported the math library to calculate the angle from this data. To smooth out the data we used the average of the last five measurements of X and Y components to calculate our angle. We also used an offset to zero the angle when the accelerometer was on the bike and on flat ground.

The gain variable, shifterGain is determined by the equation $\text{shifterGain} = \text{shifterTrim} * (1 + \sin(\theta))$. $(1 + \sin(\theta))$ is 1 on flat ground, and increases as the angle increases. The variable shifterTrim is used to tune the gain and was decided based on experimentation. In void loop(), the nominal bounds are multiplied by these gains. As the incline angle increases, the lower bound of peddling speed is increased, so the crankshaft speed will drop below the lower bound sooner. This will make the system more responsive to giving the user more torque on an incline. Of course it also optimizes the system for shifting up and speeding on a downhill.

3.2.3 Servo-motor:

To control the servo we imported the Servo library. The possible positions to move the motor to are stored in the "table" array and correspond to derailleur positions for specific gears. Each position has been calibrated by controlling the servo with a potentiometer and recording the position where the servo was in each gear.

3.3 Component Adjustment and Synergy

3.3.1 Derailleur and Servo Position

To adjust the servo positions such that the desired gear position were achieved, a separate code was used. This code used the value of a potentiometer and mapped the value to an executable servo position. The values for each trial were recorded and then saved to the gear array within the arduino code. The values for gear position were finely adjusted until optimal performance was attained.

3.3.2 Hall Effect Sensor and Voltage Drop

A small portion of code was used to record solely the pedaling RPM. The small code looked for a positive edge in the hall effect voltage data. This code was then used in the master code to prompt action. When the bank value reached the suggested limit, the servo would move to the desired position and induce a voltage drop. The voltage drop during the motion of the servo would cause an insufficient reading of the hall effect sensor and record an inaccurate reading of user pedal speed. The problem was eliminated by using two separate power supplies sharing the same ground.

4.0 Final Results

4.1 Lessons Learned

Things important to know for a group undertaking this project in the future should be aware that the biggest obstacle will be to overcome the hardware constraints imposed by the bicycle you select to use in the project. Specifically a very strong derailleur return spring will add complications to setting up a system. The stronger the return spring, the stronger of an assist spring you will have to use. These are both a hassle to work with and can end up putting strain on the servo motor if there is a situation where only one is attached for a moment.

When setting up power supplies, it is best to keep the motor power supply separate from the Arduino power supply. This is because if a single power supply is used, turning the motor can cause a big voltage drop and interfere with sensor measurements.

Finally, a complicated system with many components can be difficult to troubleshoot. Troubleshoot and debug the code for each component separately before combining them together. Also when you experience system failures, try to identify which component is giving you trouble and tinker with the base code for that component. Then introduce it into the system again. This may take several iterations and/or trying several components.

4.2 Conclusion

User reviews for our system were overwhelmingly positive. This means that Devin let several friends ride it and they all said, “awesome”. To be more specific, Devin rode the bike from his dorm in South halls to his friends’ house on South Pugh Street and back, up and down hills, speeding up and slowing down, and the system consistently delivered the appropriate gear for the situation. Don also rode the bike around campus and found that the system was very responsive. What was most surprising was that it was able to reliably downshift and give sufficient torque to the user while riding up a steep hill. Also on a flat plane the system shifted down from gear 1 to 5 fairly quickly as the user sped up. As the user slowed down, it shifted all the way back down. Overall we were very satisfied with the results and happy to learn from this project.

4.3 Arduino Code:

```
// ME445 Final Project: Automatic Bicycle Transmission
// Donald Bradfield & Devin O'Donnell

// Initializing Variables
// Performance Specifications
int SUGGEST=2; //Suggestion Level
double LB=50; //RPM
double UB=80; //RPM
double REST=25; //RPM
// Gear Calibration Values
int table[5]={0,65,82,101,150};

double lowerbound; //RPM
double upperbound; //RPM
int valold = 0; //Hall Effect previous measurement
int valnew=0; //Hall Effect current measurment
int timenew=0; //Tracking the current time measurement
int timeold=0; //Tracking the previous time measurement
double dt=0; //Change in time (Period of One Turn of Crank)
double Speed; //Rev divided by Period [rad/sec]

int DOWNBANK=0; //The value of the downbank increases if the bicycle suggests to shift
downward
int UPBANK=0; //The value of the upbank increases if the bicycle suggests to shift upward
int GEAR=3; //The initial gear of the bicycle and servo
int OLDGEAR=1; // Initialize old gear
int POS=0; //Initialize position of the servo

#include <Servo.h> //Include the servo library
Servo myservo; //Name Servo

double AcX,AcY; //Initialize Accelerometer X,Y
double acX4, acX3, acX2, acX1, acX0, acX;
double acY4, acY3, acY2, acY1, acY0, acY;

double thetaA = 0; //Angle of incline/decline [rad]
double thetaAdeg = 0; //Angle of incline/decline [deg]

double shifterGain; //The shifter gain is the value that will adjust the uppper and lower
pedelling bounds
double shifterTrim = 1; //Adjusts the shifter gain

#include<Wire.h> //Include Library for MPU6050
#include <math.h>
```



```

const int MPU=0x68; // I2C address of the MPU-6050

void setup() {
  // put your setup code here, to run once:
  myservo.attach(3);
  Wire.begin();
  Wire.beginTransmission(MPU);
  Wire.write(0x6B); // PWR_MGMT_1 register
  Wire.write(0);    // set to zero (wakes up the MPU-6050)
  Wire.endTransmission(true);
  Serial.begin(9600);
}

void loop() {
  valnew = analogRead(A0); // INPUT FROM SENSOR
  if (valold-valnew > 5){ // Looks for a Positive Edge Change from the Hall Effect Sensor
    SLOPE(); //Measure the incline to determine the shiftergain
    upperbound=shifterGain*UB; //Multiply the upper bound by the shifter gain to find the
new upper bounds
    lowerbound=shifterGain*LB; //Multiply the lower bound by the shifter gain to find the
new lower bound
    timenew=millis(); // Record the time (will be used to find the rate of the change)
    dt=timenew-timeold; // Change in time
    Serial.print(lowerbound); //Readout
    Serial.print(" < ");
    Speed=1/(dt/1000/60); // KEEP TRACK OF THE PREVIOUS MEASUREMENTS
    Serial.print(Speed);
    Serial.print(" RPM|| < "); //Readout
    Serial.print(upperbound);
    if (Speed<upperbound && Speed>lowerbound || Speed<=REST){
      Serial.print("HOLD ON!"); //Readout
      UPBANK=0;
      DOWNBANK=0;
    }
    if (Speed<lowerbound && Speed>REST){ //The user is pedaling to slow, shift down
      Serial.print("SUGGEST SHIFT DOWN"); //Readout
      UPBANK=0; //Clear upbank
      DOWNBANK++; //Add one to the downbank
    }
    if(Speed>upperbound){ //The user is pedaling to fast, shift up
      Serial.print("SUGGEST SHIFT UP "); //Readout
      DOWNBANK=0; //Clear downbank
      UPBANK++; //Add one to the upbank
    }
  }
  //GEAR_____
  Serial.print(" || "); //Readout
  Serial.print(UPBANK); //Readout
  Serial.print(" "); //Readout

```

```

    Serial.print(DOWNBANK); //Readout
    if(UPBANK==SUGGEST){ //If the upbank has reached its suggestion level, the bike
will shift upward
        UPBANK=0; //Clear bank
        GEAR++; //Shift Up
        if(GEAR>5){ //Ceiling value
            GEAR=5;
        }
    }
    if(DOWNBANK==SUGGEST){ //If the downbank reaches the suggestion limit the
bike will shift downward and clear the bank
        DOWNBANK=0; //Clear bank
        GEAR--; //Shift Down
        if(GEAR<1){ //A floor value of 1
            GEAR=1;
        }
    }
    Serial.print(" || CURRENT GEAR: "); //Readout
    Serial.println(GEAR); //Readout
}

valold=valnew; // Current value becomes old
timeold=timenew; //Current time becomes Old

    if(GEAR != OLDGEAR){ //If the Gear value has changed, the servo is prompted to
change
        POS=table[GEAR-1]; // Find the servo position with POS array
        myservo.write(POS); //Send the position to the servo
        OLDGEAR=GEAR; //Current Gear becomes old gear
        delay(1000); //Pause
    }
}

void SLOPE(){ //This function is used to track the angle of incline/decline for the bicycle
Wire.beginTransmission(MPU);
    Wire.write(0x3B); // starting with register 0x3B (ACCEL_XOUT_H)
    Wire.endTransmission(false);
    Wire.requestFrom(MPU,14,true); // request a total of 14 registers
    AcX=Wire.read()<<8|Wire.read(); // 0x3B (ACCEL_XOUT_H) & 0x3C
(ACCEL_XOUT_L)
    AcY=Wire.read()<<8|Wire.read(); // 0x3D (ACCEL_YOUT_H) & 0x3E
(ACCEL_YOUT_L)

    acX = ((acX4+acX3+acX2+acX1+acX0)/5)/1638.40;
    acY = ((acY4+acY3+acY2+acY1+acY0)/5)/1638.40;

    thetaA = atan2(acX,acY)-.2; //.2 is the bias
    thetaAdeg = (thetaA*180)/PI;

```

```
    acX4 = acX3;  
    acX3 = acX2;  
    acX2 = acX1;  
    acX1 = acX0;  
    acX0 = AcX;  
    acY4 = acY3;  
    acY3 = acY2;  
    acY2 = acY1;  
    acY1 = acY0;  
    acY0 = AcY;  
  
    shifterGain = shifterTrim*(1+sin(thetaA)); //Value to multiply bounds by  
}
```

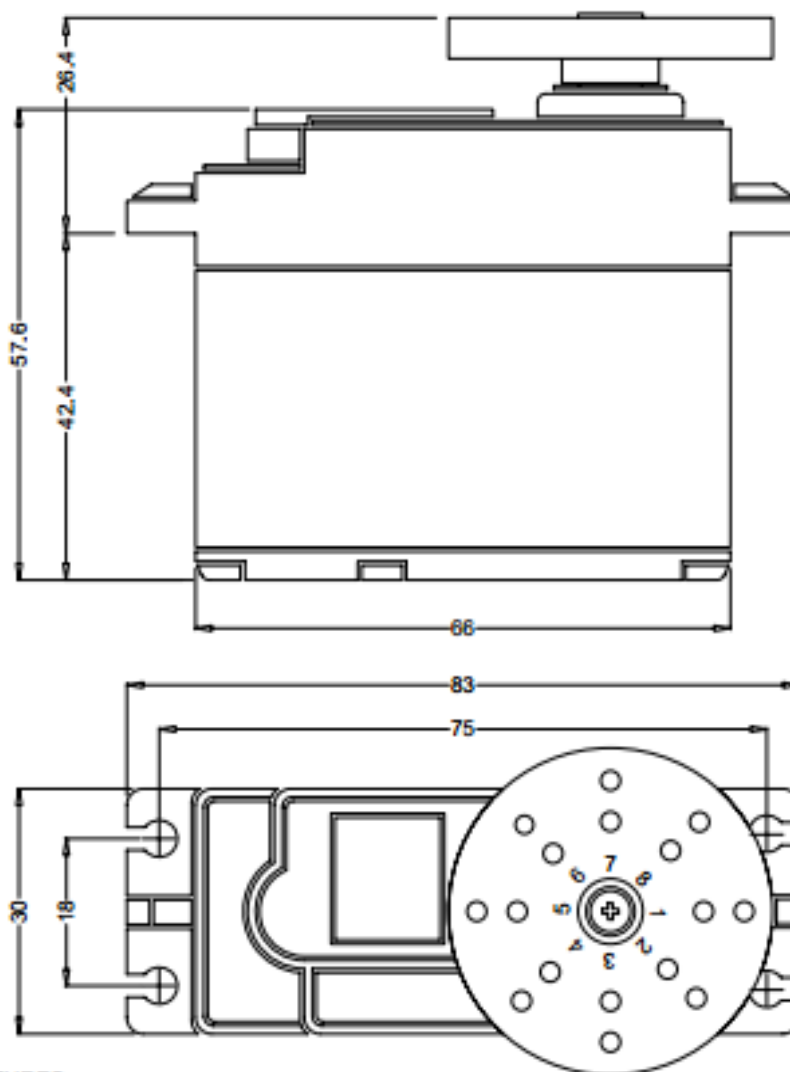
4.4 Available Datasheets for Major Components:

4.4.1 HS-805BB: <http://www.robotshop.com/media/files/pdf/hs805.pdf>

ANNOUNCED SPECIFICATION OF HS-805BB+ MEGA 1/4 SCALE SERVO

1. TECHNICAL VALUES

CONTROL SYSTEM	: +PULSE WIDTH CONTROL 1500usec NEUTRAL	
OPERATING VOLTAGE RANGE	: 4.8V TO 6.0V	
OPERATING TEMPERATURE RANGE	: -20 TO +60°C	
TEST VOLTAGE	: AT 4.8V	: AT 6.0V
OPERATING SPEED	: 0.19sec/60° AT NO LOAD	: 0.14sec/60° AT NO LOAD
STALL TORQUE	: 19.8kg.cm(274.96oz.in)	: 24.7(343.01oz.in)
OPERATING ANGLE	: 45° ONE SIDE PULSE TRAVELING 400usec	
DIRECTION	: CLOCK WISE/PULSE TRAVELING 1500 TO 1900usec	
CURRENT DRAIN	: 8mA IDLE AND 700mA/NO LOAD RUNNING	
DEAD BAND WIDTH	: 8usec	
CONNECTOR WIRE LENGTH	: 300mm(11.81in)	
DIMENSIONS	: 66x30x57.6mm(2.59x1.18x1.26in)	
WEIGHT	: 152g(5.36oz)	



2. FEATURES

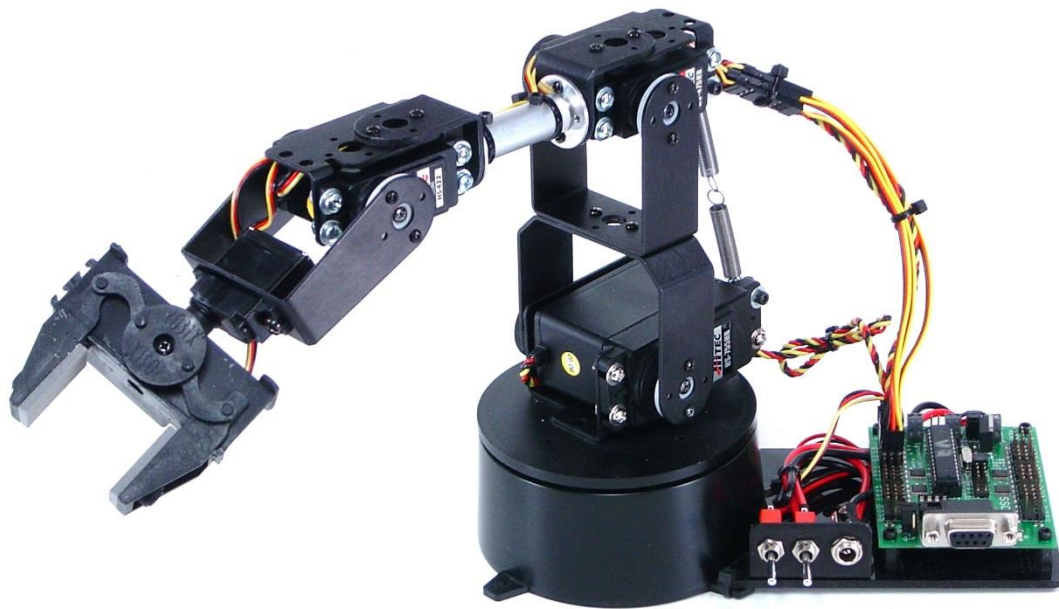
3-POLE FERRITE MOTOR
LONG LIFE POTENTIOMETER
DUAL BALL BEARING
INDIRECT POTENTIOMETER DRIVE
WATER & DUST TIGHT

3. APPLICATIONS

GIANT SCALE AIRCRAFT
1/4 SCALE CARS
LARGE SCALE BOATS

4.4.2 MPU-6050: <http://www.invensense.com/mems/gyro/documents/PS-MPU-6000A-00v3.4.pdf>

**MPU-6000 and MPU-6050
Product Specification
Revision 3.4**



CHECKER ROBOT

ME 445 Final Project

Wei-Lin Chang, Nicolas Joseph
will.chang93@gmail.com/npj5016@psu.edu

Executive Summary

This document will cover the process of designing the checker robot. The project can be split into two parts, computer part and the robot part. Computer, software, part include the derivation of the checker game, multiple methods of inputs, and calculation of the inverse kinematic for the arm movement. The hardware parts include the method of communication between Matlab, Arduino, and Pololu servo controller.

Acknowledgement

We would like to thank Mike Robinson for all the assistance that helped us through the difficulties during the project. With the help from Mike, we were able to nail down some of the toughest parts of the project. We would also like to thank Dr. Henry.J Sommer for all the intuitive lectures that helped us in the design and development process to make our project better.

Table of Contents

Introduction	4
Introduction.....	4
Bill of Material.....	4
Road map	4
Design Process.....	5
Checker Game Robot Process concept.....	5
Matlab.....	6
Matlab: Electronic Inputs	6
Matlab: Physical Inputs	7
Matlab: Game Algorithms	9
Communication between MATLAB and Arduino.....	11
Pololu Maestro Servo Controller	12
Inverse Kinematics Developments.....	13
Calculating the Base Angle.....	13
General Purpose Inverse Kinematics	14
Further Reach Inverse Kinematics	15
Conclusion and Future Improvements	15
Appendix	16
Appendix A: Matlab – Arduino_Serial	16
Appendix B: Matlab – CheckerGame.m (main function)	19
Appendix C: Matlab – Draw.....	21
Appendix D: Matlab – Exit.....	23
Appendix E: Matlab – HumanInput.....	24
Appendix F: Matlab – InitBoard.....	26
Appendix G: Matlab – InputCheck	28
Appendix H: Matlab – Movement	29
Appendix I: Matlab – OutputCheck	30
Appendix J: Matlab – Restart	32
Appendix K: Matlab – Reverse_Kin_Calc	32
Appendix L: Arduino Code	34

Introduction

Introduction

The goal of this project was to learn more about controlling a robotic arm. The project itself is a platform for a checker-playing artificial intelligence (AI) algorithm to interface the robotic arm and allow the computer to play a physical game of checkers with human users. Unfortunately due to the time constraint, we could not get an AI to function. Instead for this game a camera takes a picture to determine where the first player has moved their checker and updates the checker positions on a graphical user interface (GUI) board. The second player can then click what checker they want to move and where to move it to and the robot will move it.

The inspiration for the project came from the chess playing robot, to whom play against world chess champion Vladimir Kramnik in 2011. These robots are usually expensive and hard to obtain. When we saw the Lynxmotion AL5A robotic arm, we saw an opportunity to imitate this expensive and sophisticated robot. Due to the physical limitations, we were not able to pick up the chess pieces, thus we decided to go with a smaller and easier checker game.

In this project an Arduino Uno was used as the microcontroller for interfacing the Lynxmotion AL5A robotic arm. In order to keep wiring simple and response time adequate, we used a 6 channel Pololu Maestro servo driver to control all 6 servos. Using this servo driver allowed us to control the acceleration of the servos, which made their responds to target position more fluid-like and natural.

The majority of the details of this project lie in the software. The MATLAB code not only determined the checker board system and all of its pieces but also calculates the inverse kinematics for the servo angles to position the tip of the 5 DOF robot arm to different positions on the board. The servo angles then were communicated to the Arduino serially and the Arduino then passed a fixed sequence of moves to the servo driver.

Bill of Materials

Table 1 Bill of material for this project

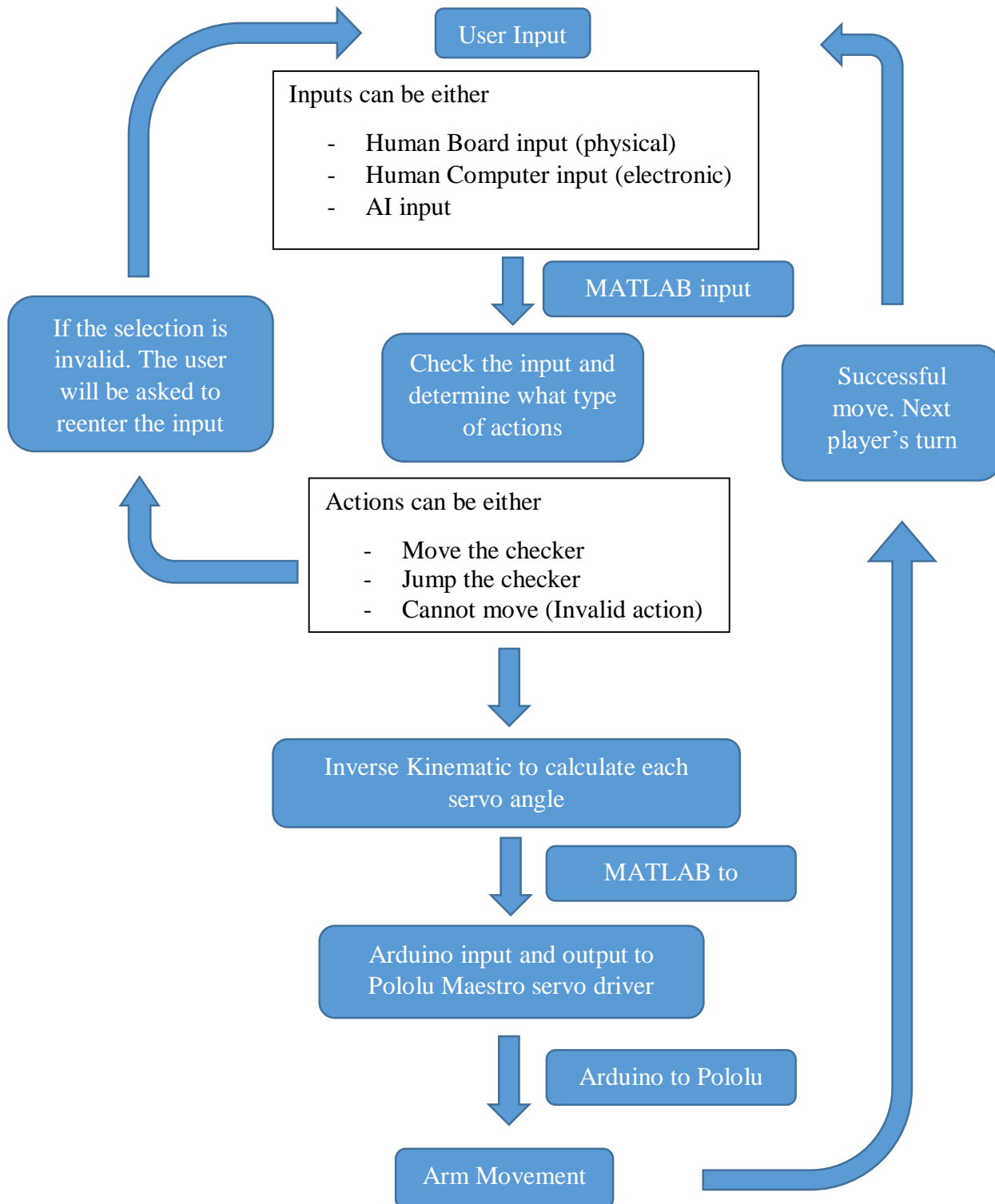
Item	Cost
Lynxmotion AL5A robotic arm combo kit	\$223.35
Medium duty wrist rotation upgrades	\$36.34
Arduino	\$29.95
Webcam	\$20.00
Pololu Maestro servo driver	\$19.95
Total	\$329.59

Road map

This report will take you through the design process of the checker robot, including the serial communication between Pololu servo controller, Arduino, and MATLAB, the process of computing the game code, the methods of collecting user inputs, and methods of moving the robot arm.

Design Process

Checker Game Robot Process concept



MATLAB

MATLAB is used as the primary software for this project which handles the main game code, inverse kinematics calculations, and the serial communication to the Arduino. Before developing a checker game robot, game rules and specifications needed to be clearly defined. The specifications and rules are as follows:

- M by N Checker board and checker
- Game rules
 - o Checkers can only move forward, unless it is a king.
 - o Checkers will upgrade to king when it reaches the end of board.
 - o Checkers can only move diagonally, cannot move vertically or horizontally.
 - o Checkers can only move one space at a time.
 - o Checkers can jump the opponents checker if it is next to it.
 - o Checkers can only jump one checker at a time.

Although there are several checker game codes on the internet to download but we had to develop our own checker game code because of the unique board size. Normal checker games have the board size of 8x8 (or 10x10 for Asian version) but our board size needed to be limited to 5x9 due to the robot's limited practical reach.

The MATLAB code can be further split into 3 sections, virtual inputs, physical inputs, and game determining algorithm.

MATLAB: Virtual Inputs

Virtual inputs, or computer inputs, are the inputs where a user uses GUI on the computer to enter the input, which was originally intended to be provided by the AI algorithm. Part of our original goal is to have an AI algorithm, so user will be able to play against computer; however, we lacked the time, resources and knowledge to finish the AI for the checker game. Developing an AI algorithm requires much skill in computer programming, which we feel would not be the responsibly of a mechanical engineer in a work environment. We ended up choosing to have on-screen GUI input method to replace the AI algorithm.

For the GUI input method, a checker board will pop up on the screen, and the user will be able to select a checker to be moved and where to move it to by clicking mouse, as shown in figure 1.

The method for determining the selection is relatively simple, it can be done by MATLAB's build in code "ginput".

```
[x,y,z] = ginput (figure#);
```

The x, y indicates the location of the selection and z indicates the button (right click v.s. left click v.s. middle click).

For detail information on the Matlab code, please refer to appendix B

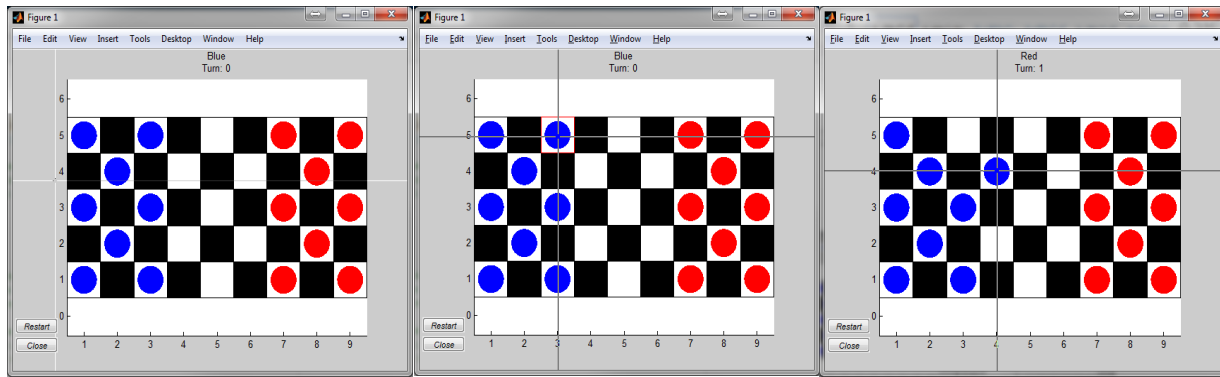


Figure 1 Checker game onscreen graphic selection method. The picture to the left is the checker board graphic. The one in the middle indicate when user selects the checker, notice the chosen checker will have a red box to confirm the selection. The picture on the right is to move the checker to the selected space. If the movement is invalid, Matlab will prompt the user to reenter the selection to move the checker to a valid location.

Matlab: Physical Inputs

The physical inputs are the most challenging part in the MATLAB game code. The ways to determine the physical inputs, including a camera (image), combination of sensors (for example sonar sensor and RGB sensor) and so on. We have decided to use the camera/ image inputs, because MATLAB came with an image processing package that can help imaging processing much easier, and also, this is the most visual method and is consider easier than other methods.

There are three main phases in determining the input, image acquiring, image processing, and determining the location/coordinates. Images are acquired by a USB webcam. In this project, the Vimicro Venus USB 2.0 PC Camera is used.

To acquire an image via webcam, a webcam support package needs to be installed. With the package, the following code can be executed to obtain the image.



Figure 2 A USB webcam. The webcam in the figure are not the same as the one used in project.

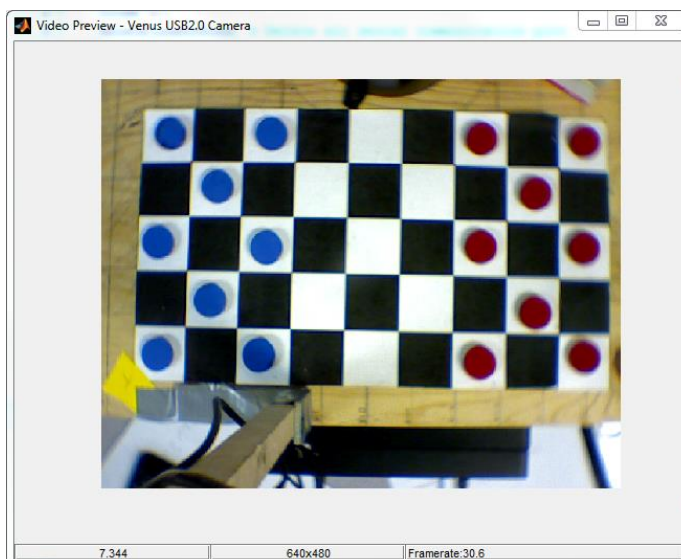


Figure 3 What Matlab see from the camera input.

```
cam = webcam;
preview(cam);
img = snapshot(cam);
```

The first code will allow variable `cam` to be assigned with the properties of the camera. Than using `preview` command will be able to see the real time live images, as shown in figure 3 to the right. The `snapshot` command will allow MATLAB to take a snapshot of the image, and store it. From the snapshots, more advance analysis and processing can be performed.

The second phase will be performing some image processing. Using some of the image processing code built in Matlab, such as

```
[imagePoints, boardSize] = detectCheckerboardPoints(img);
[Center, Radii] = imfindcircles(img,[minRadius,maxRadius],...
... 'ObjectPolarity','dark','Sensitivity',0.92,'Method','twostage');
pixels = impixel(img,Center(i,1),Center(i,2));
```

MATLAB will be able to determine where the checkers are, and what type of checkers are they.

`detectCheckerboardPoints` command will find the checker pattern on the board and output the location of the pattern as well as the size of the board, all the points are plotted in figure 4 as red circles. The blue circles in the figure are the theoretical boundary points. The reason why the theoretical points do not match with the actual board is because of the fish eye effect of the camera. Normally developers would like to correct the fish eye for a better accuracy but this step is skipped in this project.

The `imfindcircles` will find the circular objects in the picture and the `impixels` will return the color (RGB) of the point. Combining the results of the `imfindcircles` and the `impixels`, we were able to accurately plot the circles, with the same radius and color, on the figure.

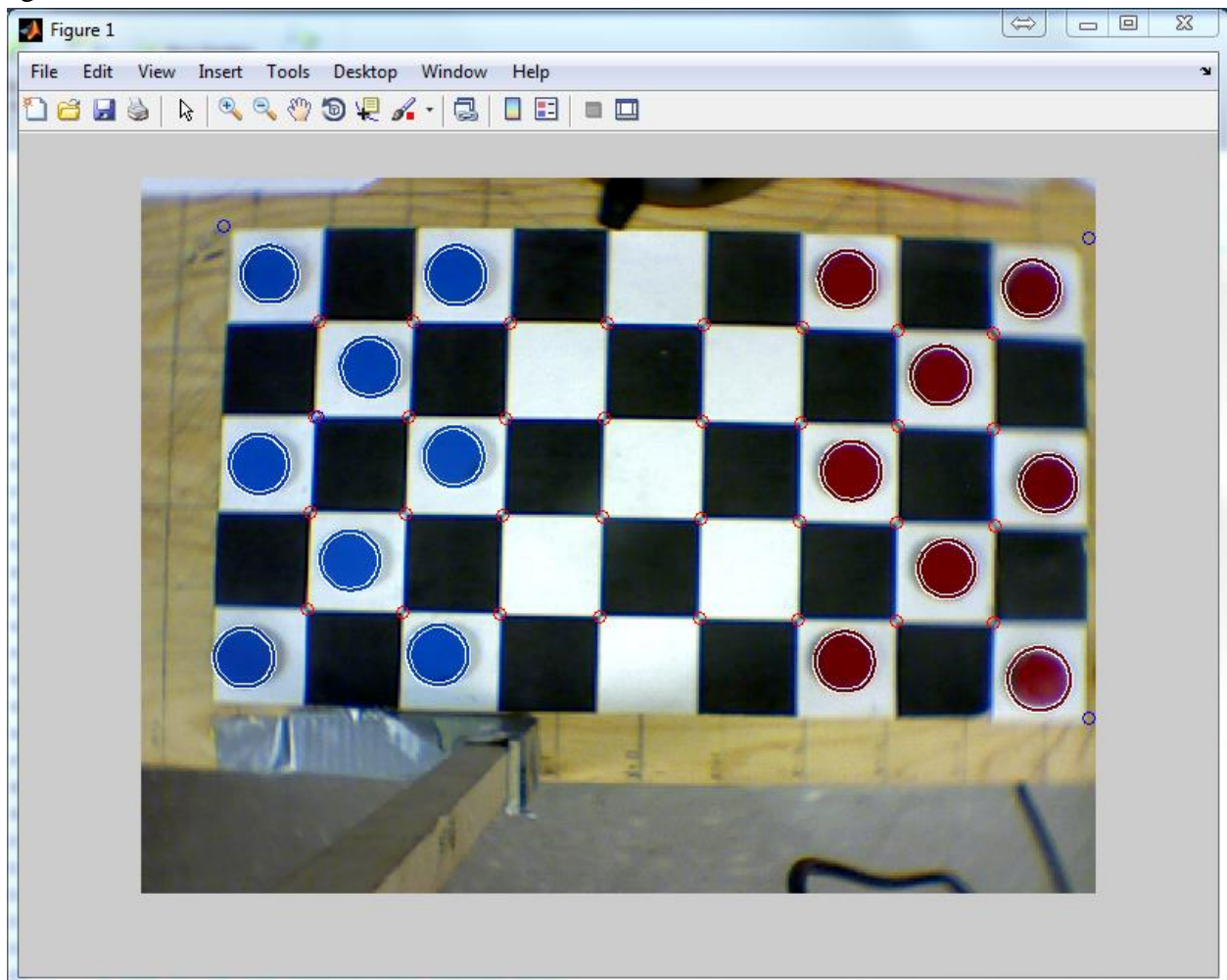


Figure 4 Matlab camera input with the processed results. The red circles are the points where a checker board pattern lies; the blue points are the theoretical checker board outer boundaries. The blue points doesn't line with the actual checker board well because of the fish eye affect. Notice the checkers are drawn with a circle with the same color, this indicates the center location, radius of the checkers, and the color of the checkers are detected correctly.

Using the locations found by the `imfindcircles`, MATLAB can determine the coordinates of the checkers on the checker board. This process involves correlations, because the locations found are in units of pixels. The correlation equations are given below.

```
x = 1 + floor((Center(i,1)-(imagePoints(1,1)-squareSize))/squareSize);
y = 5 - floor((Center(i,2)-(imagePoints(1,1)-squareSize))/squareSize);
```

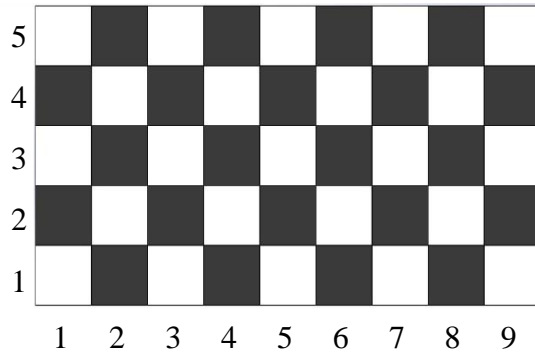


Figure 5 Coordinates system of the checker board.

To determine the input and output location of the checker, the camera simply takes a snapshot before the user moves the checker, and take another snapshot again after the user moved the checkers.

MATLAB: Game Algorithms

The game algorithms are the lengthiest part of the code. It is basically made out of bunch of if else statement that follows the rule of checker game. The game algorithms can be divided into several main parts, including input check, output check, movement, draw, and exit.

The input and output check are the algorithms that check if the user had followed the game rules, if a player has broken one of the rules, a prompt will ask the user to reenter the input/output.

The movement function will look at user's input and output to determine the action to take. There are two types of movements, jump and remove. The movement function will then replace the BoardStatus array accordingly to keep the game updated.

BoardStatus is an array that stored the current checkers on the board; including 1 (blue checkers), 2 (red checkers), and 0 (no checkers), shown in figure 5 at right.

The draw function will take the BoardStatus as an input, and generate the graphic shown in figure 1.

Current BoardStatus								
1	0	1	0	0	0	2	0	2
0	0	0	1	0	2	0	2	0
1	0	0	0	1	0	0	0	0
0	1	0	0	0	2	0	2	0
1	0	1	0	0	0	2	0	2

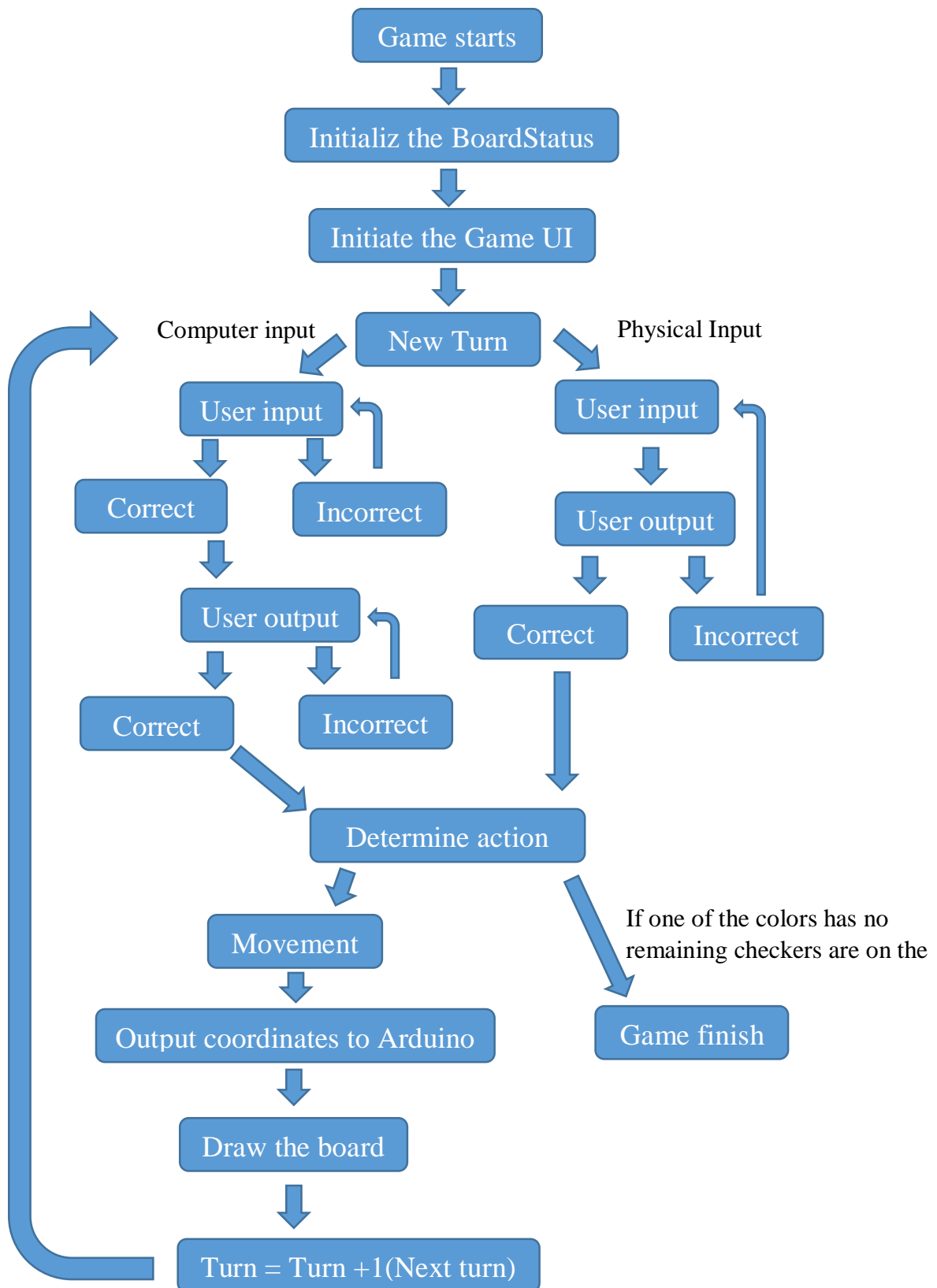
Figure 6 BoardStatus array demonstration

By pressing the “close” button, located on the left bottom of the UI, the exit function will execute. The exit function will take one last image acquiring and determine the remaining checker locations on the board. The locations will be sent to a “clean” array, which will communicate to Arduino to execute a series of moves to remove the rest of the checkers on the board.

Unfortunately due to the time constraint, the game was not complete; There were some functions missing, including upgrading from soldier to king, human error correcting and checking, or even scoring system. Although the current code is more than enough to demonstrate the ability of our checker robot, given some more time, the game can be more refine and well executed.

The process plan for the game algorithm are listed on the next page, and the code will be presented in the appendix B.

Game process plan



Communication between MATLAB and Arduino

Arduino plays a few critical roles in this system, it mainly served as a communication bridge between MATLAB and the robot arm. It receives 4 servo angles for each 3 positions via serial communication port and makes a decision on what sequence of moves to make. The MATLAB code utilizes the `fwrite()` function to send servo angles to the Arduino in the form of 8 bit unsigned integers. The first position it receives is the position of the checker to be moved, next is the location to move that checker and the third is where a checker is that needs to be removed if the robot's move jumped an opponent's checker.

If a zero value is written to the elbow and wrist for the third position, the robot will not make a third move. Also if zeros are written to the elbow and wrist of the first position, the Arduino then removes the second position from the board. This routine is used to pass positions of the checkers left on the board one by one to be cleared off the board. The reason for this particular method of communicating is to keep the same convention of sending three positions at a time to the Arduino from MATLAB. In this cleaning routine the third position is assigned an arbitrary value and is ignored. The claw is instructed to rotate based on the angle of base to keep it parallel to the length of the checker board. The only exception to this is when the arm has to reach to a far corner; in this case the claw's long face must be facing radially, or 90 degrees.

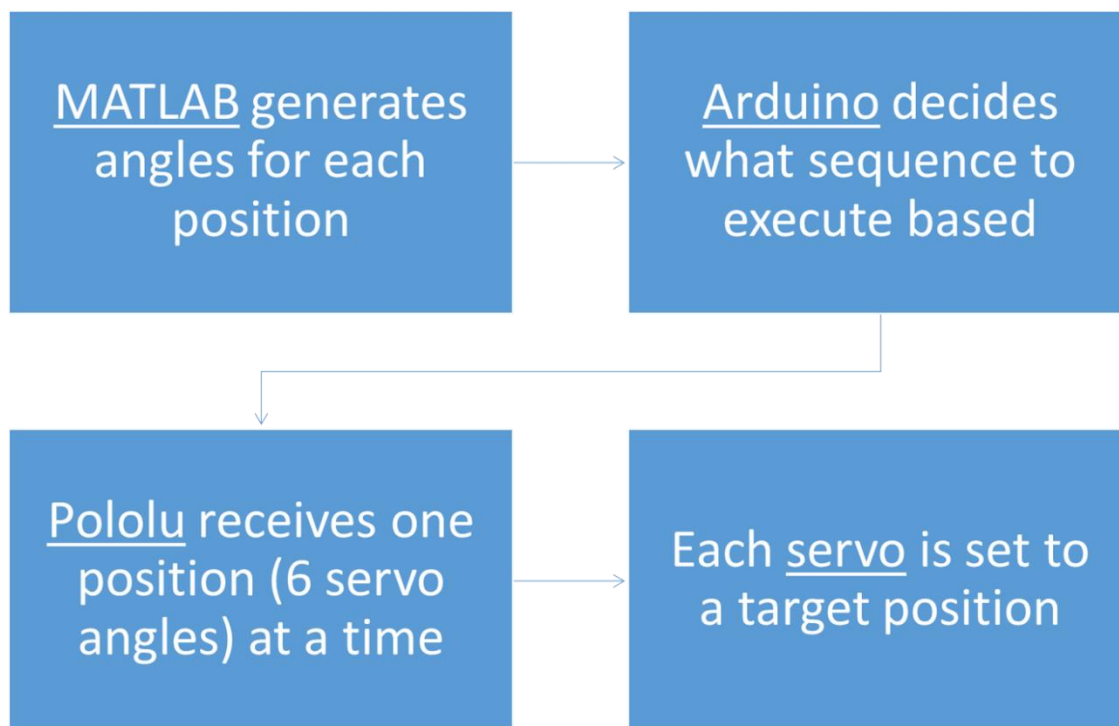


Figure 7 Matlab to physical positioning of the robotic arm.

Figure 7 illustrates the sequence taken to write the arm to a certain position

Pololu Maestro Servo Controller

The Pololu Maestro servo controller, shown in figure 8 to the right, is used in this project to reduce the complexity of controlling multiple servos. The other reason we chose to use the Pololu servo controller instead of hooking directly up to Arduino is because we observed that Arduino will act weirdly, not performing actions in sequenced as sketched, when controlling 6 servos at a time.

The first code will allow variable cam to be assigned with the properties of the camera. Then using preview command will be able to see the real time live images, as shown in figure 3 to the right. The snapshot command will allow MATLAB to take a snapshot of the image, and store it. From the snapshots, more advance analysis and processing can be performed.

The Pololu servo driver receives data bytes asynchronously via RX line in the form of 8 bits. This mean that the baud rate can be detected by the driver and no clock line is required for communication. The data format is one start bit followed by 8 data bits and a stop bit with no parity depicted in figure 9.

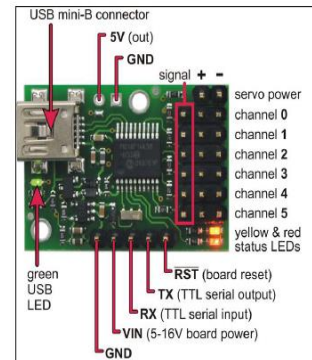


Figure 8 Pololu Maestro Servo Controller

Figure 10 shows the command protocol for writing a servo single to the servo driver. Note that the same protocol is followed to write an acceleration value to a given servo. The only difference is that there is a different command number to write acceleration .

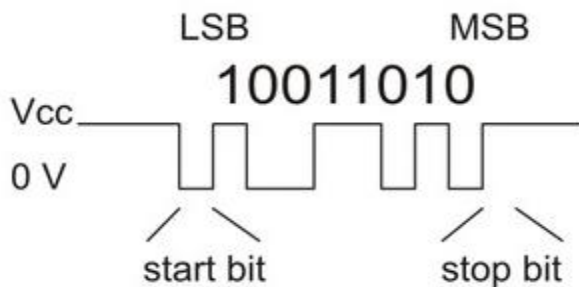


Diagram of a non-inverted TTL serial byte.

Figure 9 Courtesy of Pololu's website

```
void set_target(unsigned char servo, unsigned int target)
{
  mySerial.write(0xAA); //start byte
  mySerial.write(12); //device id (assuming it is still at the default value)
  mySerial.write(0x04); //command number with MSB cleared
  mySerial.write(servo); //servo number
  mySerial.write(target & 0x7F); // Target low byte
  mySerial.write((target >> 7) & 0x7F); // Target high byte
}
```

Figure 10 Function for writing servo angle to servo driver

Inverse Kinematics Developments

Inverse kinematics calculation is one of the most essential parts of this project. It calculates individual servo angles to reach the desire distance. The Lynxmotion AL5A robot arm has 4 degrees of freedom including wrist, elbow, shoulder and base. For the project, two inverse kinematics have been calculated, because we found one may not be able to reach all the checker coordinates.

To decrease the complexity of calculating the inverse kinematic, and to ensure less combinations of answers as possible, we incorporate some constraints into the inverse kinematics, these constraints including

- For the general purpose inverse kinematics, the claw needs to be orthogonal to the board, in favor of picking up the checkers
- For the further reach inverse kinematics, the shoulder will line up with the elbow to ensure the maximum reach.

Calculating the Base Angle

Calculating the base angle is the easiest part of the inverse kinematics. A diagram is shown to the right. The base angle can be calculate with the given equation:

$$BaseAngle = \tan^{-1} \frac{y}{x}$$

Where the x and y are the coordinate system on the checker board. These coordinates need to be correlates to length units,

and the correlations are given below

$y = abs(5 - ycoord) * 1.2$ // 1.2 is correspond to the gird size

$x = (xcoord - 1) * 1.2 + 0.6 + 0.8 + 1.9$ // the latter part is the distance from edge to base center

The base length, which will be used in later on calculation can be calculated with:

$$BaseLength = \sqrt{x^2 + y^2}$$

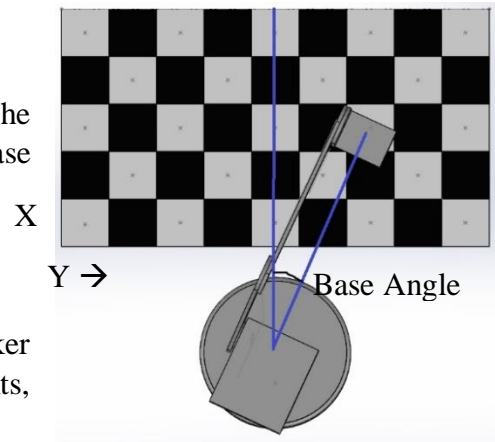


Figure 11 Diagram for calculating the base angle

General Purpose Inverse Kinematics

The general purpose inverse kinematics is the main one we use for the project. It covers up to every space, except for 2 of the further ones. A robot arm diagram is provided below to help visualizing.

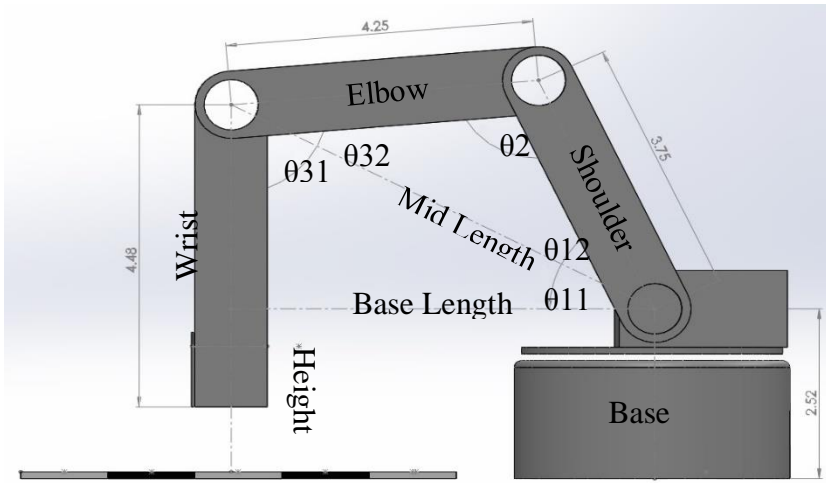


Figure 12 General purpose Inverse Kinematic diagram.

To tackle this inverse kinematics, we use the two triangles, upper ones and lower ones, between the arms. The calculation can be derived from below.

Lower triangle:

$$h = \text{Wrist} - \text{Height} + \text{desired arm height(aka z coordinates)}$$

Consequently the rest of the angles can be calculated using Pythagorean Theorem

$$\theta_{11} = \tan^{-1} \frac{h}{\text{BaseLength}}$$

$$\theta_{31} = 90 - \theta_{11}$$

$$\text{MidLength} = \sqrt{\text{BaseLength}^2 + h^2}$$

The upper triangle angles can be determined via cosine law

$$\theta_{12} = \cos^{-1} \frac{\text{Shoulder}^2 + \text{MidLength}^2 - \text{Elbow}^2}{2 * \text{Shoulder} * \text{MidLength}}$$

$$\theta_2 = \cos^{-1} \frac{\text{Shoulder}^2 + \text{Elbow}^2 - \text{MidLength}^2}{2 * \text{Shoulder} * \text{Elbow}}$$

$$\theta_{32} = 180 - \theta_{12} - \theta_2$$

Combining the results, we get:

$$\text{Shoulder Servo Angle} = \theta_1 = \theta_{11} + \theta_{12}$$

$$\text{Elbow Servo Angle} = \theta_2$$

$$\text{Wrist Servo Angle} = \theta_3 = \theta_{31} + \theta_{32}$$

Further Reach Inverse Kinematics

While the inverse kinematics above can reach most of the coordinates, there are two exceptions, coordinates (1, 1) and (1, 9); thus this inverse kinematics emphasis on the maximum outreach length.

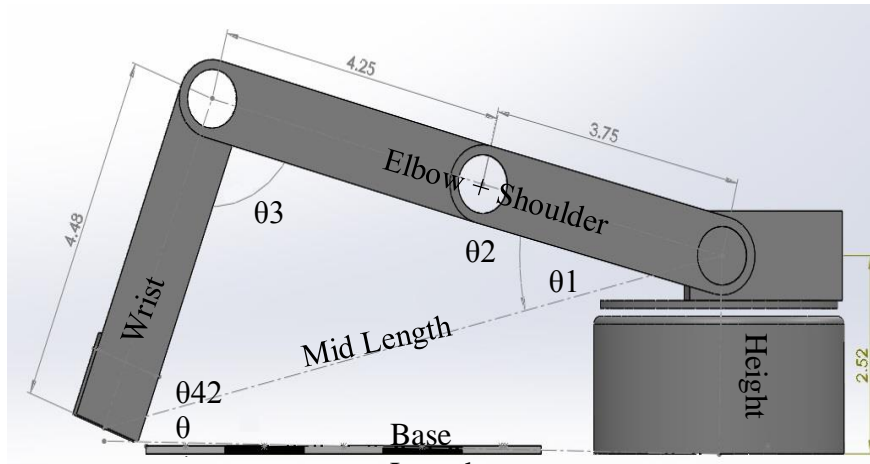


Figure 13 Further reach inverse kinematics diagram

As mentioned earlier, to maximize the outreach, the elbow and shoulder are laid parallel to one another to ensure the arm can reach to its maximum location.

The calculations are similar to the general purpose inverse kinematics.

Lower triangle

$$MidLength = \sqrt{BaseLength^2 + Height^2}$$

$$\theta_1 = \cos^{-1} \frac{(Elbow + Shoulder)^2 + MidLength^2 - Wrist^2}{2 * (Elbow + Shoulder) * MidLength}$$

$$\theta_2 = 0$$

$$\theta_3 = \cos^{-1} \frac{Wrist^2 + MidLength^2 - (Elbow + Shoulder)^2}{2 * Wrist * Midlength}$$

Conclusion and Future Improvements

In conclusion, the checker robot was a success. It had demonstrated that a relatively cheap robot arm can indeed play a checker game with human. The overall performance of the checker robot was better than what we anticipate at the beginning of this project.

Given more time, we would like to refine the robot movements, to make it more natural like, also try to add in an error correction function so the arm can pick up checkers precisely at all locations, instead of relying on checker needs to be at the center of the space. We would also like to add in some characteristics into the robot, such as, random movements or waving white flag when it loses the game. The game code itself could also be refined, as well as exploring more difference methods of physical inputs.

Appendix

Appendix A: Matlab – Arduino_Serial

```
function Arduino_Serial(Xorg,Yorg,Xtar,Ytar,Xjump,Yjump,clear)
% This function will take numbers as input and send it out to Arduino. The
% function will wait until arduino to respond before proceeding

% Updated : 12/6/14
% Updated by: Wei-Lin Chang

% Complete Set

% Last Updated : 12/13/14
% Last Updated by: Wei-Lin Chang/ Nick Joseph

% start serial connection
global s
delay=.1;

% We have to create a look up table to correct some minor drifts
% get coordinates from user, resolve angles, and send via serial  Nick
December 6, 2014
%coordinates = input('Input [x,y,z] based on grid coordinates to pick up:\n');
x_values = [-5.0 -3.7 -2.5 -1.4 -0.4 0.60 2.10 2.70 4.20          % A1 A2 A3
A4 A5 A6 A7 A8 A9
          -4.6 -3.7 -2.4 -1.4 -0.3 0.65 1.80 3.10 4.40          % B1 B2 B3
B4 B5 B6 B7 B8 B9
          -4.6 -3.6 -2.6 -1.4 -0.3 0.70 2.00 3.10 4.40          % C1 C2 C3
C4 C5 C6 C7 C8 C9
          -4.7 -3.4 -2.4 -1.4 -0.3 0.80 2.00 3.40 4.40          % D1 D2 D3
D4 D5 D6 D7 D8 D9
          -4.8 -3.5 -2.4 -1.4 -0.3 0.90 2.30 4.70 4.60];      % E1 E2 E3
E4 E5 E6 E7 E8 E9

y_values = [7.60 7.20 7.25 7.30 7.55 7.50 7.45 7.90 8.80      % A1 A2 A3
A4 A5 A6 A7 A8 A9
          6.00 6.40 6.30 6.60 6.60 6.70 6.90 6.90 7.20      % B1 B2 B3
B4 B5 B6 B7 B8 B9
          5.30 5.30 5.20 5.20 5.40 5.50 5.80 5.80 6.10      % C1 C2 C3
C4 C5 C6 C7 C8 C9
          4.25 3.90 4.00 4.00 4.30 4.40 4.60 4.50 5.20      % D1 D2 D3
D4 D5 D6 D7 D8 D9
          3.10 2.80 2.90 3.00 3.20 3.40 3.70 1.60 3.80];      % E1 E2 E3
E4 E5 E6 E7 E8 E9

z_values = [0.00 0.00 0.00 0.00 0.00 0.50 -.20 0.50 0.00      % A1 A2 A3
A4 A5 A6 A7 A8 A9
          0.00 0.20 0.00 0.00 0.00 -.10 0.00 -.05 0.00      % B1 B2 B3
B4 B5 B6 B7 B8 B9
          0.00 0.00 -.05 0.00 0.00 -.10 0.00 0.00 0.00      % C1 C2 C3
C4 C5 C6 C7 C8 C9
```

```

        0.00 -0.1 0.00 -0.1 -.10 -.10 0.00 -.10 0.00           % D1 D2 D3
D4 D5 D6 D7 D8 D9
        0.00 0.00 -0.1 0.00 -.05 0.00 0.00 0.00 0.00];       % E1 E2 E3
E4 E5 E6 E7 E8 E9

```

```

% the following remaps the indices of the checker array to the convention
% followed by Weilin's half of the code:

```

```

x_values = (fliplr(x_values));
y_values = (fliplr(y_values));
z_values = (fliplr(z_values));

```

```

% indices of position array (the zero origin corresponds to (-2.5, 2.5) on
% the physical board

```

```

orig = [Xorg,Yorg]
target = [Xtar,Ytar]
jump = [Xjump,Yjump]

```

```

if(orig(1) >0 && orig(2) >0)%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

coord_target = [ x_values(target(1),target(2)) y_values(target(1),...
    target(2)) z_values(target(1),target(2))];
[servo_target(1), servo_target(2), servo_target(3) , servo_target(4)] = ...
    Reverse_Kin_Calc( coord_target(1) ,coord_target(2), coord_target(3));

```

```

% if statement is to take care of situation where no checker has been
% jumped. Passing all zeros to arduino indicates that there is no third
% position to move to

```

```

if (( jump(1) > 0) && (jump(2) > 0))
    coord_jump = [ x_values(jump(1),jump(2)) y_values(jump(1),...
        jump(2)) z_values(jump(1),jump(2))];
    [servo_jump(1), servo_jump(2), servo_jump(3), servo_jump(4)] = ...
        Reverse_Kin_Calc( coord_jump(1) ,coord_jump(2), coord_jump(3) );
end
if ( (jump(1)==0) && (jump(2)==0))
    servo_jump(1) =0;
    servo_jump(2) =0;
    servo_jump(3) =0;
    servo_jump(4) =0;
end

```

```

coord_orig = [ x_values(orig(1),orig(2)) y_values(orig(1),...
    orig(2)) z_values(orig(1),orig(2))];
[servo_orig(1), servo_orig(2), servo_orig(3), servo_orig(4)] = ...
    Reverse_Kin_Calc( coord_orig(1) ,coord_orig(2), coord_orig(3) );

```

```

fwrite(s,servo_orig(1),'uint8')
pause(delay)
fwrite(s,servo_orig(2),'uint8')
pause(delay)
% must be unsigned integer since it ranges from 30 to 142
fwrite(s,servo_orig(3),'uint8')
pause(delay)
% must be unsigned integer since it ranges from 0 to 135

```

```

fwrite(s,servo_orig(4),'uint8')
pause(delay)

% must be unsigned integer since it ranges from 0 to 180
fwrite(s,servo_target(1),'uint8')
pause(delay)
fwrite(s,servo_target(2),'uint8')
pause(delay)
% must be unsigned integer since it ranges from 30 to 142
fwrite(s,servo_target(3),'uint8')
pause(delay)
% must be unsigned integer since it ranges from 0 to 135
fwrite(s,servo_target(4),'uint8')
pause(delay)

% must be unsigned integer since it ranges from 0 to 180
fwrite(s,servo_jump(1),'uint8')
pause(delay)
fwrite(s,servo_jump(2),'uint8')
pause(delay)
% must be unsigned integer since it ranges from 30 to 142
fwrite(s,servo_jump(3),'uint8')
pause(delay)
% must be unsigned integer since it ranges from 0 to 135
fwrite(s,servo_jump(4),'uint8')
pause(delay)
pause(4);
end

% this is the for loop to clean the board
if(orig(1) ==0 && orig(2) ==0) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for ( i = 1:(length(clean)))
    servo_orig(1) =0;
    servo_orig(2) =0;
    servo_orig(3) =0;
    servo_orig(4) =0;
    servo_jump(1) =1;
    servo_jump(2) =1;
    servo_jump(3) =1;
    servo_jump(4) =1;
% jump servo angles for cleanup are arbitrary,
% the arduino just needs a non-zero value to read

target = [clean(i,1),clean(i,2)];
coord_target = [ x_values(target(1),target(2)) y_values(target(1),...
    target(2)) z_values(target(1),target(2))];
[servo_target(1), servo_target(2), servo_target(3) , servo_target(4)] = ...
    Reverse_Kin_Calc( coord_target(1) ,coord_target(2), coord_target(3));

pause(delay)
% must be unsigned integer since it ranges from 0 to 180
fwrite(s,servo_orig(1),'uint8')
pause(delay)
fwrite(s,servo_orig(2),'uint8')
pause(delay)

```

```

% must be unsigned integer since it ranges from 30 to 142
fwrite(s,servo_orig(3),'uint8')
pause(delay)
% must be unsigned integer since it ranges from 0 to 135
fwrite(s,servo_orig(4),'uint8')
pause(delay)

% must be unsigned integer since it ranges from 0 to 180
fwrite(s,servo_target(1),'uint8')
pause(delay)
fwrite(s,servo_target(2),'uint8')
pause(delay)
% must be unsigned integer since it ranges from 30 to 142
fwrite(s,servo_target(3),'uint8')
pause(delay)
% must be unsigned integer since it ranges from 0 to 135
fwrite(s,servo_target(4),'uint8')
pause(delay)

% must be unsigned integer since it ranges from 0 to 180
fwrite(s,servo_jump(1),'uint8')
pause(delay)
fwrite(s,servo_jump(2),'uint8')
pause(delay)
% must be unsigned integer since it ranges from 30 to 142
fwrite(s,servo_jump(3),'uint8')
pause(delay)
% must be unsigned integer since it ranges from 0 to 135
fwrite(s,servo_jump(4),'uint8')
pause(delay)
pause(4);
%end
end

end

```

Appendix B: Matlab – CheckerGame.m (main function)

```

clear all
close all
delete(instrfind) % Delete all serial communication port
clc

% Start the Serial Communication
s = serial('COM31', 'BaudRate', 9600);
% COM port will vary from computer to computer
set(s, 'InputBufferSize', 512);
fopen(s);

global s
global BoardStatus turn action
global xin yin zin xout yout zout

```



```

global Cposy Cposx Cv
% Global variable name and function
% Turn: how many turns in the game. Even is blue, and Odd is red.
% BoardStatus: this will give the status of the entire board.
% Number Checker Status
% 1 blue solider
% 2 red solider
% 3 blue king
% 4 red king
% xin, yin, zin: from the graphic input. Checker selection
% xout, yout, zout: from the graphic input. Checker move-about
% Cposx, Cposy, Cv: Where are the blue and red checkers

% Local Variables name and function
% Conti = 0(No) / 1(Yes), if to continue the game, the game will determine
% once one of the gamer wins, or gamer terminate the game
Conti = 1;

% Initalizing the board
InitBoard(); % This will create the initial conditions for the game
turn = 0;
% creating buttons, these buttons serve as interupts
uicontrol('Style','pushbutton','string','Close',...
          'Callback','Exit','Position',[5,25,60,20],...
          'FontAngle','italic')
uicontrol('Style','pushbutton','string','Restart',...
          'Callback','Restart','Position',[5,50,60,20],...
          'FontAngle','italic')

while Conti == 1
    clc
    % Initial declariation
    % Display board status(For troubleshooting purpose)
    disp('Current BoardStatus')
    for i = 5:-1:1
        disp(BoardStatus(i,:))
    end
    % Display title
    if mod(turn,2) == 0 % Even, Blue
        title({'Blue', ['Turn: ', num2str(turn)]})
    elseif mod(turn,2) == 1 % Odd, Red
        title({'Red', ['Turn: ', num2str(turn)]})
    end

    if mod(turn,2) == 0 % Even, Blue, Robot's turn
        % Input Check
        permission = 0;
        % The user won't be able to move on untile permission is 1
        % which mean valid moves
        while permission == 0
            [xin,yin,zin] = ginput(1);
            xin = round(xin);
            yin = round(yin);
            [Cposy,Cposx,Cv] = find(BoardStatus);

```

```

        permission = InputCheck();
        if permission == 0
            disp('Invalid Selection, Please Try Again')
        end
    end

    % Output Check
    permission = 0;
    while permission == 0
        [xout,yout,zout] = ginput(1);
        xout = round(xout);
        yout = round(yout);
        if zout == 3
            break;
        end
        permission = OutputCheck();
        if permission == 0
            disp('Invalid Move, Please Try Again')
        end
    end
elseif mod(turn,2) == 1 % Odd, Red, Human's turn
    permission = 0;
    while permission == 0
        % Missing correcting and error check function for human input
        HumanInput()
        permission = InputCheck();
        permission = OutputCheck();
        if permission == 0
            % A pop-out windows
            uiwait(msgbox('Please Put The Checker Back To The Original
Place','Fail!!!!!!','warn')));
        end
    end
end

if permission == 1
    Movement()
    Draw(BoardStatus)
    turn = turn + 1;
end
% ressetting the color
rectangle('Position',[xin-0.5,yin-0.5,1,1],'edgecolor','k');

disp('Now Translating to Arduino')
% Pause to let the arm finish moving.
% Given more time, this can be change to wait until arudino send back a
% serial input
% while s.BytesAvailable == 0
% end
pause(3);
end

```

Appendix C: Matlab – Draw

```

function Draw(BoardStatus)
%{

```

This function will draw the board. It takes the BoardStatus as input and will draw the checkers accordingly to the boardstatus. The 1 will be drawn blue, 2 will be drawn red, and 0 will be drawn white.

```
% Last Update Date: 11/26/14
% Last Update by: Wei-Lin Chang
%}

% Draw entire board

clf
figure(1)
% to ensure the buttons stays on the graph
uicontrol('Style','pushbutton','string','Close',...
          'Callback','Exit','Position',[5,25,60,20],...
          'FontAngle','italic')
uicontrol('Style','pushbutton','string','Restart',...
          'Callback','Restart','Position',[5,50,60,20],...
          'FontAngle','italic')
% Draw Checker Board
for ii = 0.5:8.5
    axis equal
    for jj = 0.5:4.5
        hold on
        rectangle('Position',[ii jj 1 1],'linewidth',0.5,'edgecolor','k');
        X=[0+ii 1+ii 1+ii 0+ii];
        Y=[0+jj 0+jj 1+jj 1+jj];
        if mod(mod(ii,2)+mod(jj,2),2) == 0
            fill(X,Y,'k');
        else
            fill(X,Y,'w');
        end
    end
end

% Draw "Checkers" onto the board
[y,x] = size(BoardStatus);
for i = 1:x
    for j = 1:y
        % No checker and white box
        if BoardStatus(j,i) == 0 && mod(i,2) == 0 && mod(j,2) == 0
            Xcir = i + 0.4*cosd(1:360);
            Ycir = j + 0.4*sind(1:360);
            fill(Xcir,Ycir,'w','EdgeColor','w');
        elseif BoardStatus(j,i) == 0 && mod(i,2) == 1 && mod(j,2) == 1
            Xcir = i + 0.4*cosd(1:360);
            Ycir = j + 0.4*sind(1:360);
            fill(Xcir,Ycir,'w','EdgeColor','w');
        elseif BoardStatus(j,i) == 1 % Blue Checker
            Xcir = i + 0.4*cosd(1:360);
            Ycir = j + 0.4*sind(1:360);
            fill(Xcir,Ycir,'b','EdgeColor','w');
        elseif BoardStatus(j,i) == 2 % Red Checker
            Xcir = i + 0.4*cosd(1:360);
            Ycir = j + 0.4*sind(1:360);
            fill(Xcir,Ycir,'r','EdgeColor','w');
```

```

        end
    end
end

end

```

Appendix D: Matlab – Exit

```

function Exit()
% This function will Exit the game
% Before exiting the game. The robot will clean out all the remaining
% checkers on the board.
% The remaining checker on the board will be determined by the Camera,
% using the same process as obtaining the human input

% For the detail of each line, refer to HumanInput.m
% Initializing the camera
clear('cam');
delete('cam');
cam = webcam;
pause(2);
img = snapshot(cam);

% Calculating the grid size/ checker radius
[imagePoints, boardSize] = detectCheckerboardPoints(img);
squareSize = sqrt((imagePoints(1,1)-imagePoints(2,1))^2 ...
    + (imagePoints(1,2)-imagePoints(2,2))^2);
squareSize = round(squareSize);

minRadius = round(squareSize/2*0.55);
maxRadius = round(squareSize/2*0.7);

% Obtaining the locations of the remaining checkers
img = snapshot(cam);
[Center, Radii] =
imfindcircles(img, [minRadius, maxRadius], 'ObjectPolarity', 'dark', 'Sensitivity',
    0.92, 'Method', 'two-stage');

% Obtaining the coordinates of the remaining checkers
for i = 1:size(Center)
    if Center(i,1) ~= 0 && Center(i,2) ~= 0
        clean(i,2) = floor((Center(i,1)-(imagePoints(1,1)-
squareSize))/squareSize)+1; % x
        clean(i,1) = 5 - floor((Center(i,2)-(imagePoints(1,1)-
squareSize))/squareSize); % y
    end
end
end
Arduino_Serial(0,0,2,2,3,3,clean);
clear('cam')
clear
close all

end

```

Appendix E: Matlab – HumanInput

```
function [ ] = HumanInput()
% This function will activate the camera and ask for human input
% Last Update date: 12/10/14
% Last Update By: Wei-Lin Chang

global BoardStatus turn action
global xin yin zin xout yout zout
global Cposy Cposx Cv

%% Initialize the camera
% Setup the camera
cam = webcam;
% Start a new windows to display real time image from the camera
%preview(cam);
% pause for 2 second to allow the shutter to adjust
% to prevent from getting images that are not bright enough
pause(2)

%% Initialize the checker board

HumanBoardStatusOld = zeros(5,9);
HumanBoardStatusNew = zeros(5,9);
%% Image Gathering

% Take a Snapshot of the currently viewing image
img = snapshot(cam);
% Show the image; Not Necessary
%{
figure(2) % So no conflict with figure(1) drawing board
imshow(img)
%}

%% Image processing (board)
% Find the checker pattern and determine imagepoints
[imagePoints, boardSize] = detectCheckerboardPoints(img);
% use Pythagorean rule  $c^2 = a^2 + b^2$  to find the grid size
squareSize = sqrt((imagePoints(1,1)-imagePoints(2,1))^2 ...
    + (imagePoints(1,2)-imagePoints(2,2))^2);
squareSize = round(squareSize);

%{
Grid Size = 1.2", and checkers are roughly .75" in diameter,
so is roughly 62.5% of the grid size.
%}
minRadius = round(squareSize/2*0.55); % Estimate the minimum radius that may
occur
maxRadius = round(squareSize/2*0.7); % Estimate the minimum radius that may
occur

%% Image processing (user input)

% Take a Snapshot of the currently viewing image
img = snapshot(cam);
```

```

% This will find the circles on the image
[Center, Radii] =
imfindcircles(img, [minRadius, maxRadius], 'ObjectPolarity', 'dark', 'Sensitivity'
, 0.92, 'Method', 'twostage');

% For each center point, find the color at the point, and determine if the
% checker is blue or red
for i = 1:size(Center)
    pixels = impixel(img, Center(i,1), Center(i,2));
    if pixels(1) > pixels(2) && pixels(1) > pixels(3) % Red
        x = floor((Center(i,1)-(imagePoints(1,1)-squareSize))/squareSize)+1;
        y = 5 - floor((Center(i,2)-(imagePoints(1,1)-squareSize))/squareSize);
        % replace the location on the old board
        HumanBoardStatusOld(y,x) = 2;
    elseif pixels(3) > pixels(2) && pixels(3) > pixels(1) % Blue
        x = floor((Center(i,1)-(imagePoints(1,1)-squareSize))/squareSize)+1;
        y = 5 - floor((Center(i,2)-(imagePoints(1,1)-squareSize))/squareSize);
        % replace the location on the old board
        HumanBoardStatusOld(y,x) = 1;
    end
end

clc
disp('Current Board')
for i = 5:-1:1
    disp(HumanBoardStatusOld(i,:))
end

%% Image processing (user input)

disp('Please Make Your Movements')

uiwait(msgbox({'Click Okay When You Decided Your Move', 'Think Carefully
Before You Move'}, 'Its Your Turn', 'warn'));

% Take a Snapshot of the currently viewing image
img = snapshot(cam);
[Center, Radii] =
imfindcircles(img, [minRadius, maxRadius], 'ObjectPolarity', 'dark', 'Sensitivity'
, 0.92, 'Method', 'twostage');
for i = 1:size(Center)
    pixels = impixel(img, Center(i,1), Center(i,2));
    if pixels(1) > pixels(2) && pixels(1) > pixels(3) % Red
        x = floor((Center(i,1)-(imagePoints(1,1)-squareSize))/squareSize)+1;
        y = 5 - floor((Center(i,2)-(imagePoints(1,1)-squareSize))/squareSize);
        HumanBoardStatusNew(y,x) = 2;
    elseif pixels(3) > pixels(2) && pixels(3) > pixels(1) % Blue
        x = floor((Center(i,1)-(imagePoints(1,1)-squareSize))/squareSize)+1;
        y = 5 - floor((Center(i,2)-(imagePoints(1,1)-squareSize))/squareSize);
        HumanBoardStatusNew(y,x) = 1;
    end
end

clc
disp('Current Board')

```

```

for i = 5:-1:1
    disp(HumanBoardStatusNew(i,:))
end

%% Coordinates Check
clc

[OX,OY,OV] = find(HumanBoardStatusOld);
[NX,NY,NV] = find(HumanBoardStatusNew);
count = 0;

% Found that if a checker is being eaten, than this checking sequence may
% not work
for i = 1:size(OX)
    for k = 1:size(OX)
        if OX(i) == NX(k)
            if OY(i) == NY(k)
                if OV(i) == NV(k)
                    % replace the checker status to null
                    HumanBoardStatusOld(OX(i),OY(i)) = 0;
                    HumanBoardStatusNew(NX(k),NY(k)) = 0;
                    count = count +1;
                    %break;
                end
            end
        end
    end
end
end

[OX,OY,OV] = find(HumanBoardStatusOld);
[NX,NY,NV] = find(HumanBoardStatusNew);

% The Xin/outs numbsers might need to be adjusted
% as the matrix is actually showing reversed status of the board

xin = OY;
yin = OX;
xout = NY;
yout = NX;
fprintf('Input(%0.0f,%0.0f), and Output(%0.0f,%0.0f)\n',xin,yin,xout,yout)

%%
clear('cam')

end

```

Appendix F: Matlab – InitBoard

```
function [] = InitBoard()
```

```

% This function will create the initial condition of the cheker board.
% Including
% - Drawing the initial "Board"
% - Create the initial condition
% Update Date: 11/20/14
% Update by: Wei-Lin Chang

% - Draw the Checkers
% - Add in global variables
% Last Update Date: 11/26/14
% Last Update by: Wei-Lin Chang

global BoardStatus

clf
figure(1);

% This part will create the checker board figure it self
for ii = 0.5:8.5
    axis equal
    for jj = 0.5:4.5
        hold on
        rectangle('Position',[ii jj 1 1],'linewidth',0.5,'edgecolor','k');
        X=[0+ii 1+ii 1+ii 0+ii];
        Y=[0+jj 0+jj 1+jj 1+jj];
        if mod(mod(ii,2)+mod(jj,2),2) == 0
            fill(X,Y,'k');
        else
            fill(X,Y,'w');
        end
    end
end

% This part will create the initial checker status
% BoardStatus is a variable that will keep track all the checkers and its
% locations. The blue checker will appear as 1, and red as 2. 0 will be the
% spaces that is not occupied by any checker

% Place Blue Checkers
for r = 1:5
    for c = 1:3
        if mod(r,2) == mod(c,2) || mod(r,2) == mod(c,2)
            BoardStatus(r,c) = 1;
        end
    end
end
% Place Empty Spaces
for r = 1:5
    for c = 4:6
        if mod(r,2) == mod(c,2) || mod(r,2) == mod(c,2)
            BoardStatus(r,c) = 0;
        end
    end
end
% Place Red Checkers
for r = 1:5

```



```

    for c = 7:9
        if mod(r,2) == mod(c,2) || mod(r,2) == mod(c,2)
            BoardStatus(r,c) = 2;
        end
    end
end
end

```

```

Draw(BoardStatus)

```

Appendix G: Matlab – InputCheck

```

function [permission] = InputCheck()
% This function will take the user inputs and check if they have selected
% the correct checker or not.
% For most of the time, 0(even) turns will be blue, and 1(odd) turns will
% be red.

% Update Date: 11/20/14
% Update by: Wei-Lin Chang

% - Add in the for i = 1: size statement
%
% Last Update Date: 11/26/14
% Last Update by: Wei-Lin Chang

global turn
global xin yin
global Cposy Cposx Cv

permission = 0;

if mod(turn,2) == 0 % Even, Blue
    for i = 1:size(Cposy)
        if Cv(i) == 1 && xin == Cposx(i) && yin == Cposy(i)
            permission = 1;
            rectangle('Position',[xin-0.5,yin-0.5,1,1],'edgecolor','r');
        end
    end
elseif mod(turn,2) == 1 % Odd, Red
    for i = 1:size(Cposy)
        if Cv(i) == 2 && xin == Cposx(i) && yin == Cposy(i)
            permission = 1;
            rectangle('Position',[xin-0.5,yin-0.5,1,1],'edgecolor','r');
        end
    end
end
end
end

```

Appendix H: Matlab – Movement

```
function Movement()
%{
This function will take the action as input, and perform the corresponding
movement
- 1: no action, can't move
- 2: can jump to the location.
- 3: Eat the Checker

Updated date: 11/29/14
Updated by: Wei-Lin Chang

- Explanation changed

Last update: 12/6/14
Last update by: Wei-Lin Chang

%}
global BoardStatus turn action
global xin yin zin xout yout zout
global Cposy Cposx Cv

switch action
    case 1 % Move the checker
        % Initial block to null; (moved away)
        BoardStatus(yin,xin) = 0;
        % Replace the block by corresponding checker
        if mod(turn,2) == 0
            BoardStatus(yout,xout) = 1;
        elseif mod(turn,2) == 1
            BoardStatus(yout,xout) = 2;
        end
        % for V1 version (No Camera Input)
        % Arduino_Serial(yin,xin,yout,xout,0,0,[]);
        if mod(turn,2) == 0 % Even, Blue
            Arduino_Serial(yin,xin,yout,xout,0,0,[]);
        end
    case 2 % Eat the checker
        % Initial block to null; (moved away)
        BoardStatus(yin,xin) = 0;
        % Taken away the eaten checker; (eat out)
        BoardStatus((yin+yout)/2,(xin+xout)/2) = 0;
        % Replace the block by corresponding checker
        if mod(turn,2) == 0
            BoardStatus(yout,xout) = 1;
        elseif mod(turn,2) == 1
            BoardStatus(yout,xout) = 2;
        end
        % for V1 version (No Camera)
        % Arduino_Serial(yin,xin,yout,xout,(yin+yout)/2,(xin+xout)/2,[]);
        if mod(turn,2) == 0 % Even, Blue
            Arduino_Serial(yin,xin,yout,xout,(yin+yout)/2,(xin+xout)/2,[]);
        end
    end
end
end
```

Appendix I: Matlab – OutputCheck

```
function [permission] = OutputCheck()
%{
This fuction will take the graphic input from the user, and check is the
movement is valid. There are several rules
- The soliders can only move forward.
- All checkers can only move to the same color board.

Last Update Date: 11/27/14
Last Update by: Wei-Lin Changs
%}

global BoardStatus turn action
global xin yin zin xout yout zout
global Cposy Cposx Cv

permission = 0;
ocp = 0;
action = 0;

if mod(turn,2) == 0 % even turn, blue checker
    friendly = 1; % Same Color checker
    rival = 2; % Different Color Checker
elseif mod(turn,2) == 1 % odd turn, red checker
    friendly = 2;
    rival = 1;
end

% This version will only check the around the selected version
% Same line, can't not move
if xout-xin == 0
    action = 0;
    permission = 0;

% abs(xout-xin) == abs(yout-yin) == 1, this won't work in this case because
% the == between two numbers will split out a true-false(1-0).
elseif abs(xout-xin) == 1 && abs(yout-yin) == 1
    % within 1 blcok radius
    for i = 1:size(Cv)
        if xout == Cposx(i) && yout == Cposy(i)
            % The block is ocupied by any color
            action = 0;
            permission = 0;
            break;
        else
            % No occupaion, proceed to move
            action = 1;
        end
    end
end

elseif abs(xout-xin) == 2 && abs(yout-yin) == 2
    % within 2 blcok radius
    % need two separte for loop, because the one might occur
    % earlier than another one
    for i = 1:size(Cv)
```

```

        if xout == Cposx(i) && yout == Cposy(i)
            % The block is occupied by any color
            action = 0;
            permission = 0;
            ocp = 1;
            break;
        end
    end
end
for i = 1:size(Cv)
    if (xout+xin)/2 == Cposx(i) && (yout+yin)/2 == Cposy(i) && ocp ~= 1
        if Cv(i) == friendly
            % Can't jump over a friend
            action = 0;
            permission = 0;
            break;
        elseif Cv(i) == rival
            % Jump and eat rival
            action = 2;
            break;
        end
    else
        % There are no checker in between, can't jump
        action = 0;
        permission = 0;
    end
end
end
else
    % Invalid selection
    action = 0;
    permission = 0;
end
end

```

% This switch action will make sure the checkers will be move within the
% limits of the board.

```

switch action
    case 1 % Moving Checker
        if mod(turn,2) == 0 % even turn, blue checker
            if xout == xin + 1 && xout <= 9 && xout >= 1
                if yout == yin + 1 && yout <= 5
                    permission = 1;
                elseif yout == yin - 1 && yout >= 1
                    permission = 1;
                end
            end
        elseif mod(turn,2) == 1 % odd turn, red checker
            if xout == xin - 1 && xout <= 9 && xout >= 1
                if yout == yin + 1 && yout <= 5
                    permission = 1;
                elseif yout == yin - 1 && yout >= 1
                    permission = 1;
                end
            end
        end
    end
case 2 % Eating the Checker
    if mod(turn,2) == 0 % even turn, blue checker

```

```

        if xout == xin + 2 && xout <= 9 && xout >= 1
            if yout == yin + 2 && yout <= 5
                permission = 1;
            elseif yout == yin - 2 && yout >= 1
                permission = 1;
            end
        end
    elseif mod(turn,2) == 1 % odd turn, red checker
        if xout == xin - 2 && xout <= 9 && xout >= 1
            if yout == yin + 2 && yout <= 5
                permission = 1;
            elseif yout == yin - 2 && yout >= 1
                permission = 1;
            end
        end
    end
end
end
end
end

```

Appendix J: Matlab – Restart

```

function Restart()
% Restart the game

```

```

clc
clear

```

```

close all

```

```

CheckerGame

```

```

end

```

Appendix K: Matlab – Reverse_Kin_Calc

```

function [baseangle,shoulderangle,elbowangle,wristangle] =
Reverse_Kin_Calc(xcoord , ycoord, zcoord)
% REVERSE_KIN_CALC Summary of this function goes here
% This function will take the x and y coordinates of the game piece, and
% turn it into 4 servo angle position.

```

```

% Update date: 12/6/14
% Update by: Wei-Lin Chang

```

```

% Discarded(!!!)

```

```

% - Code

```

```

% Update date: 12/7/14
% Update by: Wei-Lin Chang

```

```

% - If Statement

```

```

% Last update date: 12/7/14

```

```

% Last update by: Nick

% - minor changes
% - Second inverse Kin

% Last update date: 12/16/14
% Last update by: Nick

% Name Conversation
% -----
% Baseangle = base plate servo angle
% Shoulderangle = theta1
% elboangle = theta2
% wristangle = theta 3
% Shoulder = R1
% elbow = R2
% wrist = R3

% Constants
% -----
Shoulder = 3.625; % R1
Elbow = 4.25; % R2
Wrist = 4.5; % R3
Baseheight = 2.5 + 0.3; %(height + clearance(prevent from hitting board))

x = xcoord;
y = ycoord;
z = zcoord+Wrist-Baseheight;

% Inverse Kin
% Input x,y,z coordinates, output servo angles

% the alternative to this is a far reach
if (sqrt((ycoord.^2) + (xcoord.^2))<9) %(ycoord < 7.6) && (abs(xcoord) <5)
% Calculate the base parameters
ThetaB = atan2d(y,x);
baseangle = ThetaB;
if (ThetaB <0)
    baseangle = ThetaB +180;
end

% Calculating the Lengths
Baselength = sqrt(x^2+y^2); % dw(G) on calculation paper
Midlength = sqrt(Baselength^2+z^2); % l(G) on calculation paper

% Lower Triangle
Theta12 = acosd(Baselength/Midlength);
Theta32 = 90 - Theta12;

% Upper Triangle
Theta2 = abs(acosd((Shoulder^2+Elbow^2-Midlength^2)/(2*Shoulder*Elbow)));
Theta11 = abs(acosd((Shoulder^2+Midlength^2-Elbow^2)/(2*Shoulder*Midlength)));

```

```

Theta31 = abs(acosd((Midlength^2+Elbow^2-Shoulder^2)/(2*Midlength*Elbow)));

% Final Angles
Theta1 = Theta11+Theta12;
Theta3 = Theta31+Theta32;

% Servo outputs
% Servo 1 Base Servo
% if xcoord>5
%     baseangle = 90 + ThetaB;
% elseif xcoord<5
%     baseangle = 90 - ThetaB;
% elseif xcoord == 5
%     baseangle = 90;
% end
% Servo 2 Shoulder Servo
shoulderangle = Theta1;
% Servo 3 Elbow Servo
elbowangle = Theta2 - 15;% -15 => Angle Calibration
% Servo 4 Wrist Servo
wristangle = Theta3 - 90;

else %if checker is beyond normal reach of robot (radius)
    z = zcoord-Baseheight;
    elbowangle = 180;
    ThetaB = atan2d(y,x);
    baseangle = ThetaB;
    if (ThetaB <0)
        baseangle = ThetaB +180;
    end

    Baselength = sqrt(x^2+y^2); % dw(G) on calculation paper
    Midlength = sqrt(Baselength^2+z^2); % l(G) on calculation paper
    wristangle = acosd(((Shoulder+Elbow)^2 + Wrist^2 -
Midlength^2)/(2*(Shoulder+Elbow)*Wrist)) -90;
    shoulderangle = acosd(((Shoulder+Elbow)^2 - Wrist^2
+Midlength^2)/(2*(Shoulder+Elbow)*Midlength))-atan2d(Baseheight,Baselength);

end
end

```

Appendix L: Arduino Code

```
#include <SoftwareSerial.h>
```

```
#define txPin 4
```

```
#define rxPin 3
```

```
// servo numbers
```

```
int base = 3; //0
```

```

int shoulder = 0; // 1
int elbow = 1; // 2
int wrist = 2; // 3
int rotate = 4;
int claw = 5;
int clawopen = 4500;
int clawclose = 6500;

// Arrays where all of the servo angles are stored
double servoOrig[4]; // Location of checker to be moved
double servoTarget[4]; // Target location for checker move
double servoJump[3]; // if robot jumps an opponents checker, this is the location of
that checker

SoftwareSerial mySerial(rxPin, txPin);
void setup() {
// put your setup code here, to run once:
// open serial connection and flush out any floating data
mySerial.begin(9600);

Serial.begin(9600);
Serial.flush();
delay(500);

// set position to the initial position
set_target(base,map(180,180,0,1984,10240));
delay(2);
set_target(shoulder,map(90,0,110,2600,6528));
delay(2);
set_target(elbow,map(75,142,30,3200,8000));

```



```
delay(2);
set_target(wrist,map(0,0,135,2000,7784));
delay(2);
set_target(rotate,map(180,180,0,1984,10240));//set_target(rotate,map(90,0,90,5100,9000));
delay(2);
// set claw open
set_target(claw,clawopen);
delay(500);
}
void loop() {
// put your main code here, to run repeatedly:
// waits for the buffer to fill with 5 values
// 4 values for servos, 1 value for pick up or drop off order
while(Serial.available()<12)
{
// reads in 4 servo angles and 1 gripper status
servoOrig[0] = Serial.read();
delay(11);
servoOrig[1] = Serial.read();
delay(11);
servoOrig[2] = Serial.read();
delay(11);
servoOrig[3] = Serial.read();
servoOrig[4] = 0;
delay(11);
servoTarget[0] = Serial.read();
delay(11);
servoTarget[1] = Serial.read();
```

```
delay(11);
servoTarget[2] = Serial.read();
delay(11);
servoTarget[4] = 0;
servoTarget[3] = Serial.read();
delay(11);
servoJump[0] = Serial.read();
delay(11);
servoJump[1] = Serial.read();
delay(11);
servoJump[2] = Serial.read();
delay(11);
servoJump[3] = Serial.read();
delay(11);
```

```
// first moves shoulder and elbow upwards to avoid knocking over object
set_target(shoulder,map(90,0,110,2600,6528));
delay(2);
set_target(elbow,map(90,142,30,3200,8000));
delay(500);
```

```
// During normal operation these servo angles will be >0, if they are not the robot will follow an
else statement
```

```
// causing it to wait for the positions to clear the board
```

```
if ((servoOrig[4] > 0) && ( servoOrig[2] >0)){
```

```
// FIRST POSITION
```

```
// sets base anand wrist, and pauses before lowing the elbow and setting shoulder
```

```
// to avoid hitting the object
```

```
set_target(base,map(servoOrig[0],180,0,1984,10240));
```

```

delay(2);
if (servoOrig[2] ==180){
  set_target(rotate,map(90,180,0,1984,10240));
}
else{
  set_target(rotate,map(servoOrig[0],180,0,1984,10240));
}
delay(2);
set_target(wrist,map(servoOrig[3],0,135,2000,7784));
delay(750);
set_target(elbow,map(servoOrig[2],142,30,3200,8000));
delay(2);
set_target(shoulder,map(servoOrig[1],0,110,2600,6528));
delay(1000);
set_target(claw, clawclose);
delay(400);
////////////////////////////////////
// first moves shoulder and elbow upwards to avoid knocking over object
set_target(shoulder,map(90,0,110,2600,6528));
delay(2);
set_target(elbow,map(90,142,30,3200,8000));
delay(500);
// SECNOND POSITION
// sets base anand wrist, and pauses before lowing the elbow and setting shoulder
// to avoid hitting the object
set_target(base,map(servoTarget[0],180,0,1984,10240));
delay(2);

```

```

if (servoTarget[2] ==180){
  set_target(rotate,map(90,180,0,1984,10240));
}
else{
  set_target(rotate,map(servoTarget[0],180,0,1984,10240));
}

delay(2);
set_target(wrist,map(servoTarget[3],0,135,2000,7784));
delay(750);
set_target(elbow,map(servoTarget[2],142,30,3200,8000));
delay(2);
set_target(shoulder,map(servoTarget[1],0,110,2600,6528));
delay(1000);
set_target(claw, clawopen);
delay(400);
////////////////////////////////////
// first moves shoulder and elbow upwards to avoid knocking over object
set_target(shoulder,map(90,0,110,2600,6528));
delay(2);
set_target(elbow,map(90,142,30,3200,8000));
delay(500);

if ((servoJump[4] > 0) && ( servoJump[2] >0)){
  // THIRD POSITION
  // sets base anand wrist, and pauses before lowing the elbow and setting shoulder
  // to avoid hitting the object
  set_target(base,map(servoJump[0],180,0,1984,10240));

```

```

delay(3);

if (servoJump[2] == 180){
  set_target(rotate, map(90, 180, 0, 1984, 10240));
}
else{
  set_target(rotate, map(servoJump[0], 180, 0, 1984, 10240));
}

delay(2);
set_target(wrist, map(servoJump[3], 0, 135, 2000, 7784));
delay(750);
set_target(elbow, map(servoJump[2], 142, 30, 3200, 8000));
delay(2);
set_target(shoulder, map(servoJump[1], 0, 110, 2600, 6528));
delay(1000);
set_target(claw, clawclose);
delay(400);
}

////////////////////////////////////

//return to origin
set_target(shoulder, map(90, 0, 110, 2600, 6528));
delay(2);
set_target(elbow, map(75, 142, 30, 3200, 8000));
delay(2);
set_target(wrist, map(0, 0, 135, 2000, 7784));
delay(300);
set_target(base, map(180, 180, 0, 1984, 10240));

```

```

delay(2);
set_target(rotate,map(90,180,0,1984,10240));//set_target(rotate,map(90,0,90,5100,9000));
delay(2);
// setgripper open
delay(1000);
set_target(claw,clawopen);
delay(500);
}

else{ //Cleanup - where target and jump positions then become positions of checkers to pick up
set_target(base,map(servoTarget[0],180,0,1984,10240));
delay(2);

if (servoTarget[2] ==180){
    set_target(rotate,map(90,180,0,1984,10240));
}
else{
set_target(rotate,map(servoTarget[0],180,0,1984,10240));
}

delay(300);
set_target(wrist,map(servoTarget[3],0,135,2000,7784));
delay(750);
set_target(elbow,map(servoTarget[2],142,30,3200,8000));
delay(2);
set_target(shoulder,map(servoTarget[1],0,110,2600,6528));
delay(650);
set_target(claw, clawclose);

```

```

delay(500);

//return to origin
set_target(shoulder,map(90,0,110,2600,6528));
delay(2);
set_target(rotate,map(90,180,0,1984,10240));//set_target(rotate,map(90,0,90,5100,9000));
delay(2);
set_target(elbow,map(75,142,30,3200,8000));
delay(2);
set_target(wrist,map(0,0,135,2000,7784));
delay(300);
set_target(rotate,map(90,0,180,1984,10240));//set_target(rotate,map(90,0,90,5100,9000));
delay(50);
set_target(base,map(180,180,0,1984,10240));
delay(2);
// setgripper open
delay(1000);
set_target(claw,clawopen);
delay(400);
}
}

// function for set_target
void set_target(unsigned char servo, unsigned int target)
{

if(servo <4){ // Writes the accleration for smooth transistions
mySerial.write(0xAA); //start byte

```

```
mySerial.write(12); //device id (the default value)
mySerial.write(0x09); //command number with MSB cleared
mySerial.write(servo); //servo number
mySerial.write(12 & 0x7F); // acceleration low bit acceration of 12
mySerial.write((24 & 0x7F)); //acceleration high bit
}
mySerial.write(0xAA); //start byte
mySerial.write(12); //device id (assuming it is still at the default value)
mySerial.write(0x04); //command number with MSB cleared
mySerial.write(servo); //servo number
mySerial.write(target & 0x7F);
mySerial.write((target >> 7) & 0x7F);
}
```


ME 445 Final Report

Automatic Bike Camera

Daniel Cho
Justin Kinslow

12/19/2014

Table of Contents

1. Introduction	2
2. Hardware	3
2.1 Bill of Materials.....	3
2.2 Component Overview	3
Camera	3
SD Card.....	4
XBee Module.....	4
Ultrasonic Sensor	5
Servo Motor	5
Hall Effect Sensor	6
LED Display	6
3. Software	7
3.1 Arduino Uno	7
3.2 RedBot	8
4. Device Operation	9
5. Results and Discussion	10
6. Conclusion.....	12
Appendix A: Code.....	13
Arduino Code	13
RedBot Code	16

All spec sheets are included at the end.

1. Introduction

The goal of this project was to design a system that would catch violators of the the Pennsylvania statute 75 Pa.C.S. § 3303 that says that drivers must maintain at least a four foot zone between them and a biker on the side of the road. The driver would be caught by using a camera mounted on a servo to take a picture of the car's license plate. This picture would then be stored to an SD card which could be removed later to view the pictures taken on the ride.

The distance between the car and the bike was found using an ultrasonic sensor mounted onto a RedBot board from Sparkfun. When the ultrasonic sensor was tripped, it would send a signal to turn the servo and then it would communicate with the Arduino Uno board via an XBee module to take the picture. Additionally, a Hall Effect sensor and a magnet were used to calculate the speed of the bike. This value was then printed onto an LED readout to give a digital speedometer. It was originally intended to use this speed with the second ultrasonic sensor to get the cars speed relative to the bike. The relative speed would then be used to change the rate that the servo would turn.

The motivation behind this project was to create a safer experience for bikers. We were interested in doing something with a bike at the beginning of the semester, and after talking to our friends that were frequent bikers we determined that one of the largest issues affecting bikers in State College was drivers getting too close to them on main roads. The idea behind the camera is twofold. It is designed to both catch the violator in the act while creating a system that will discourage drivers from breaking this law because there is a way to record them doing it. This camera module could be compacted and sold as a product to enhance overall biker safety.

2. Hardware

2.1 Bill of Materials

Table 1: Bill of Materials

Quantity	Vendor	Component	Price
1x	Adafruit	TTL Serial JPEG Camera	\$39.95
1x	Adafruit	MicroSD Card Breakout	\$14.95
1x	Adafruit	Quad Alpha-Numeric Display – Red	\$9.95
2x	Sparkfun	Ultrasonic Sensor	\$25.95
2x	Sparkfun	XBee	\$24.95
1x	Sparkfun	RedBot	\$49.95
1x	Arduino	Uno	\$24.56
1x	Jameco	Hall-Effect Sensor	\$0.89
1x	Jameco	Servo Motor	\$14.95
Total			\$206.10

Most of the components used in this project were found in the lab or provided. The only purchased parts in the final design were the camera, microSD card reader, and the LED display.

2.2 Component Overview

Camera

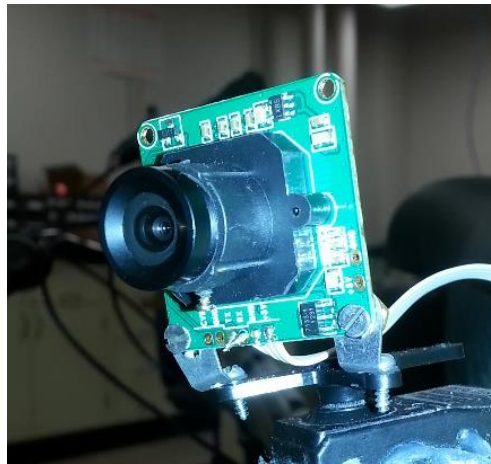


Figure 1: Camera

The purpose of the camera is to take a picture through a command from the Arduino. The camera would rotate at a certain angle from the servo motor and then snap a picture of a license plate and then return to its original position.

SD Card

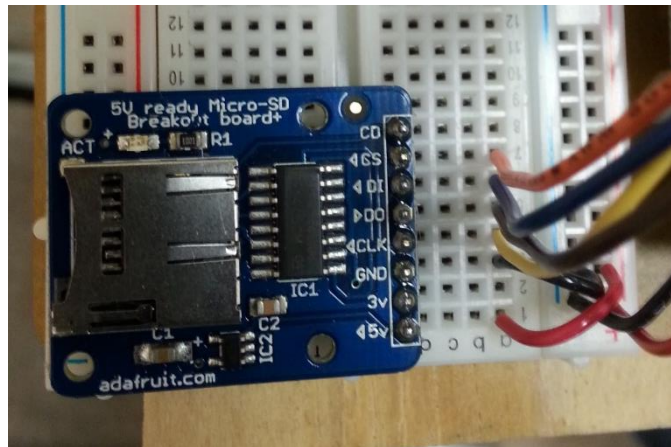


Figure 2: SD Card Reader

The MicroSD Card Breakout reads a microSD card and then stores information. It will would store the image that was captured from the camera. Then, the stored image would be transferred onto a computer. One of the main disadvantages to this reader is that it takes a long time to physically store the picture.

XBee Module



Figure 3: XBee

The XBee is a module that can gather, store and transfer data through Wi-Fi. The XBee module is used to communicate between the Arduino and the RedBot. The Uno will receive a signal from the RedBot that tells the camera to take a picture.

Ultrasonic Sensor



Figure 4: MaxBotix Ultrasonic Sensor

The MaxBotix sensor is able to measure the distance to an object. There are several ways you can wire it to do this, but for our project we used the analog readings. The device will return an analog voltage, and we used the code uploaded to the RedBot to convert this into a usable distance. This device has very good resolution, especially outdoors but is susceptible to giving false reading from time to time.

Servo Motor

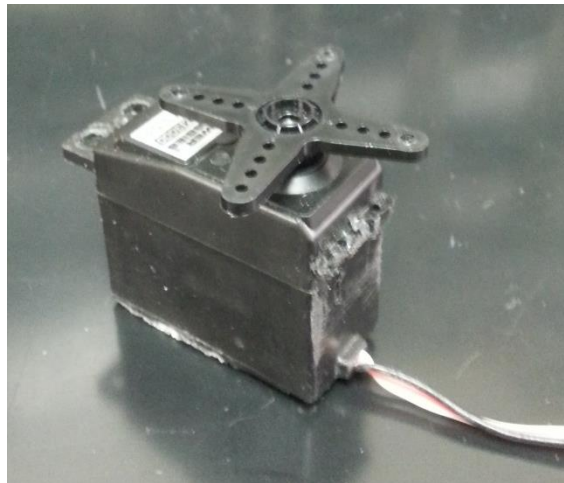


Figure 5: Servo Motor

The Servo Motor is a DC motor that can rotate to a certain angle. The camera is mounted on top of the servo motor and then rotates at a certain angle to capture the violator's license plate. The servo was an excellent choice for this application because it is easy to control the rotation rate and fit directly onto the RedBot board.

Hall Effect Sensor



Figure 6: Hall Effect Sensor

The Hall-Effect Sensor is a transducer that varies its output signal in response to a magnetic field. The sensor is used to measure the rotation of the front wheel of the bike through a magnetic. This value is then translated in the speed at which the bike is travelling.

LED Display

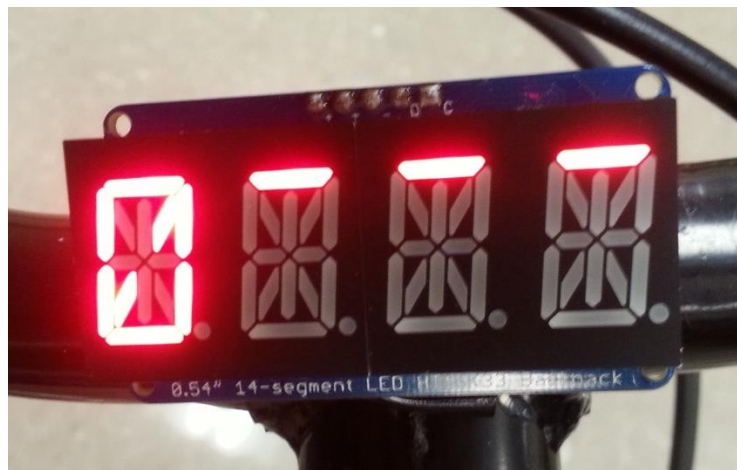
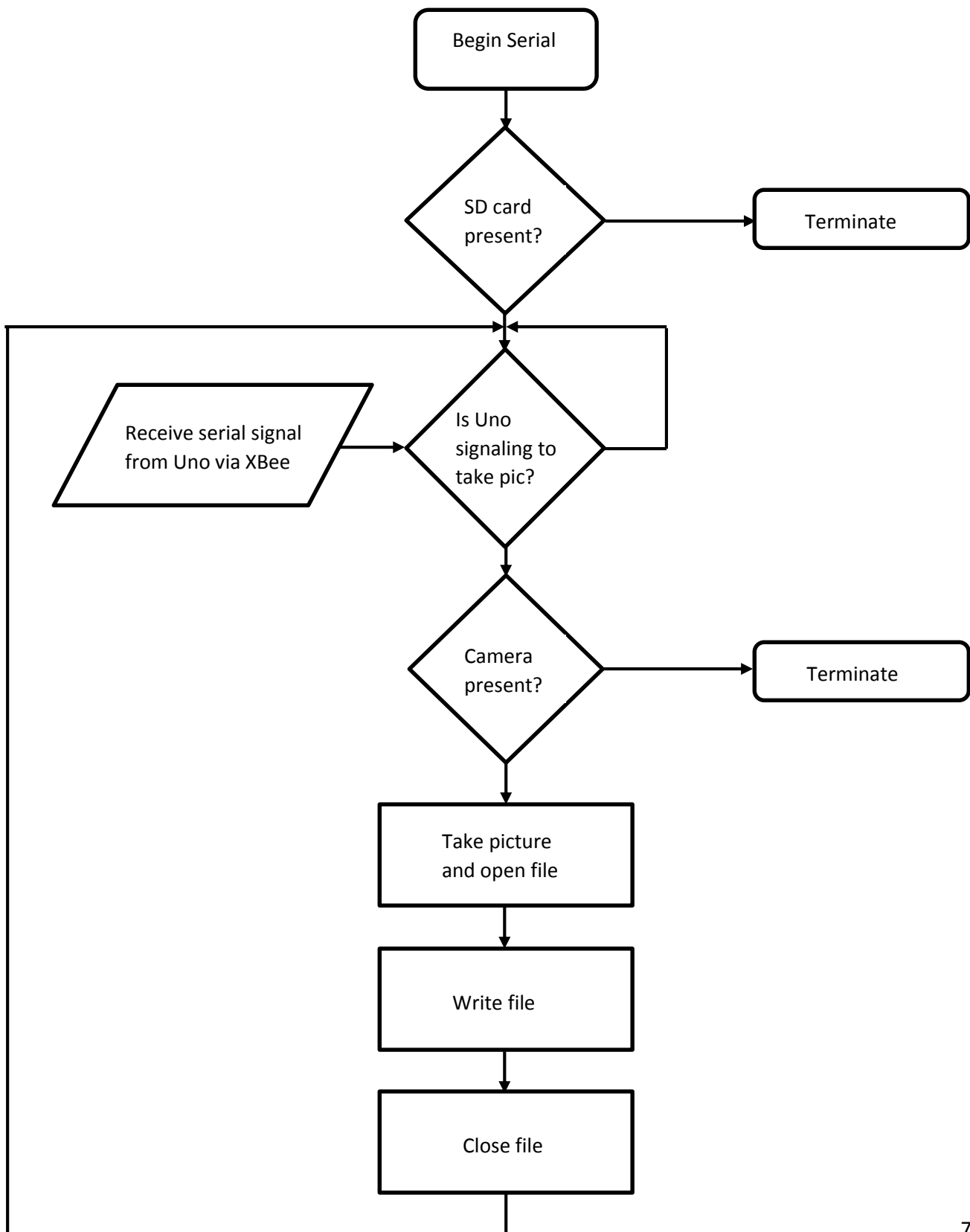


Figure 7: LED Alphanumeric Display

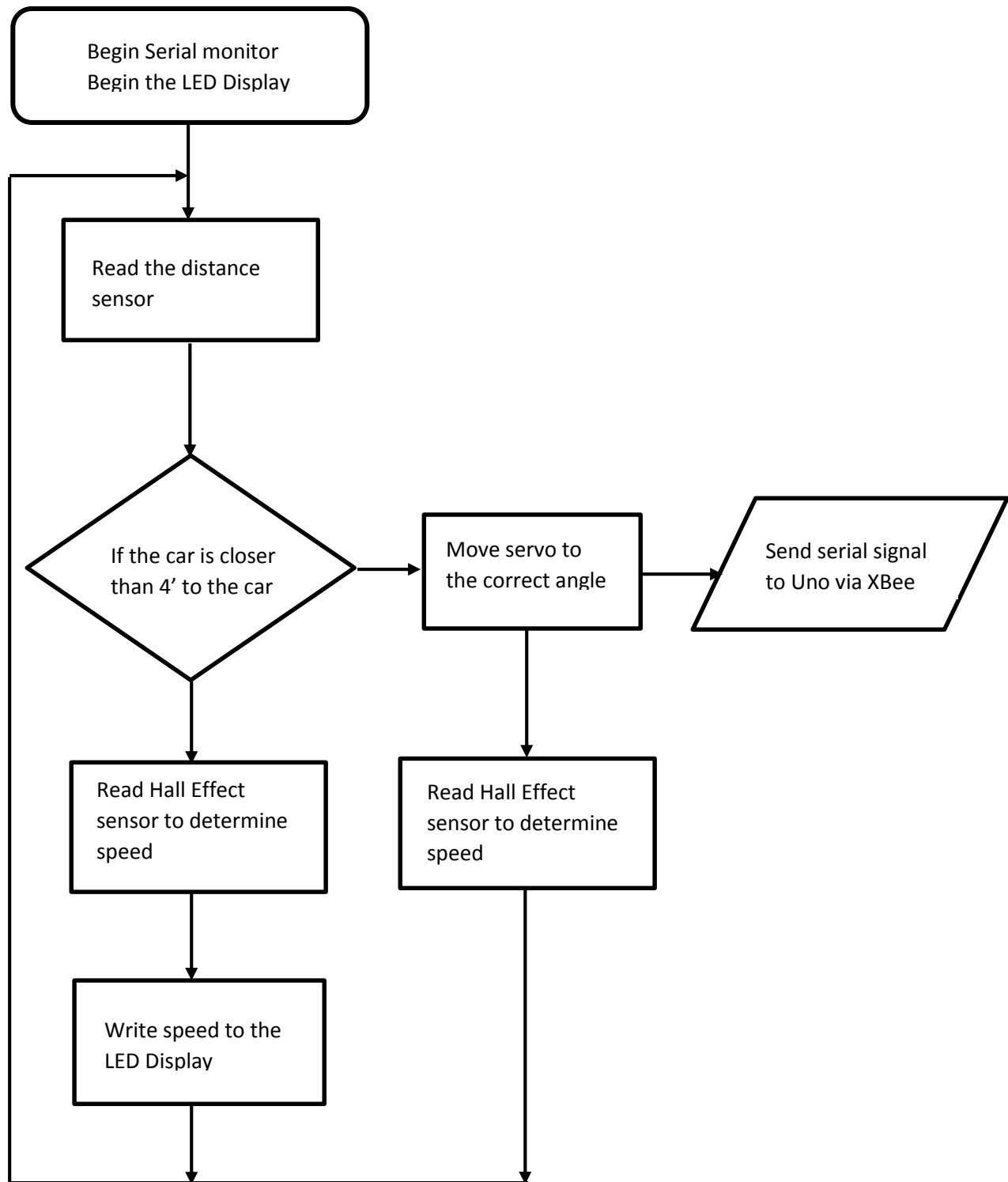
The LED displays the speed the bike is going. The RedBot relies on I2C communication with the driver chip to control all of the displays at once. However, the software for this device was difficult to program, and therefore the LED display only prints out an integer value rather than a more exact speed.

3. Software

3.1 Arduino Uno



3.2 RedBot



4. Device Operation

The Arduino UNO controls the camera and the SD card reader and the RedBot will control the ultrasonic sensor and the servo motor. These two boards will communicate with each other via the XBee modules. The RedBot will be coded to trigger the camera and the servo motor by giving a buffer distance of 4'. It means that if an object, in this case a car, is within the 4' distance then the sensor will trigger the servo motor to rotate at a given angle. Then the Redbot will send a signal to trigger the camera from the UNO. The camera will have a delay until the servo reaches its position and then will take a picture of the license plate. Then, the UNO will signal the microSD Card Breakout to store the snapped image to the inserted microSD card. The UNO will then communicate with the RedBot's servo motor to return to its original position.

Additionally, a Hall Effect sensor is being used to read a change in magnetic field from the bike wheel. The sensor is mounted on the front wheel frame and the magnetic is mounted on one of the wheel spoke. When the bike is in motion, the magnet triggers the sensor. The LED display then shows the speed that the bike is moving. Both of these components are controlled by the RedBot.

Finally, in order to transfer the stored picture, the SD card has to be inserted in an adapter then the adapter has to be inserted to a computer. Once the computer reads the adapter, the picture can be transferred onto the computer and read.



Figure 8: Final Bike Assembly

5. Results and Discussion

After developing code that worked for each of the components, we combined the code that would be uploaded onto both the Arduino Uno and RedBot respectively. We assembled a mounting system and began to troubleshoot the entire system. Minor issues were fixed relatively quickly, but there were several issues that took a good deal of troubleshooting.

One of the biggest issues we had was getting the camera to take and store a new picture each time. The camera would go through the loop and write a picture to the SD card each time, but the image saved to each new photo would always be the one taken first. For example, if the camera took images 1, 2, and 3, it would store images 1, 2, and 3 but every image would be the same as image 1. To fix this we had to begin the camera each time it was triggered to take a picture, which added to the processing time but removed the issue.

The largest issue that we were unable to resolve was using two sensors in tandem. With this setup, one sensor would read the distance the car is away from the bike and one sensor would calculate the speed at which the car was moving. Using the speed from the Hall Effect sensor, we planned on calculating the relative speed of the car and using it to control the rotation rate of the servo. The issue with using two separate sensors was that the operation of one would interfere with the other and create a huge deal of noise. To overcome this, we eventually figured out a way to clock the sensors to read them at the same time. We were able to test this on a breadboard, but made the decision to not implement this setup and focus on just using one sensor well instead.

The completed project, while missing some of our intended components, did well when testing on the road. However, some small issues still remain. One is that the sensor will randomly trip due to noise or a person will activate it rather than a car. Additionally, we found that while the focus on the camera performed well indoors, it needed to be adjusted for outside performance. Despite some of these issues, the project was able to perform and get a picture of a car that violated the statute as shown in figure.



Figure 9: Result from Road Testing

6. Conclusion

Overall we deemed this project a success. The code and hardware combined to work exactly as intended and we were able to get a picture of a license plate of the car that violated the statute during our road testing. There are still some small issues, such as the false positive readings or the focus of the camera, but these are simple fixes to implement.

If we were to begin this project again with what we know now, we would have gone about several things differently. First of all, we would have implemented the other sensor to add the ability to sense the cars relative speed. We also would have gone with an LCD readout rather than the LED display we were working with due to the LCD's increased versatility. Finally, we would have looked at a more efficient way to store and display the pictures taken during operation.

Appendix A: Code

This appendix contains the final code that the team members developed for both the Arduino Uno and the RedBot board.

Arduino Code

```
//JEK$DC
//controls camera, writes images to SD card
//Update: 12/18/2014
//XBee serial info: XBeel, 13A20040E7F8AD

#include <Adafruit_VC0706.h>
#include <SD.h>
#include <SoftwareSerial.h>

//SD card chip select
//Adafruit SD shields and modules: pin 10
#define chipSelect 10

//camera TX connected to pin 2, camera RX to pin 3:
SoftwareSerial cameraconnection = SoftwareSerial(2, 3);

Adafruit_VC0706 cam = Adafruit_VC0706(&cameraconnection);

void setup() {
  pinMode(10, OUTPUT); //SD card pin as output

  Serial.begin(9600); //begin serial
  Serial.println("VC0706 camera snapshot");

  //see if the card is present and can be initialized:
  if (!SD.begin(chipSelect)) {
    Serial.println("Card failed, or not present");
    // don't do anything more:
    return;
  }
}

void loop() {
  int getData = Serial.read(); //read serial, RedBot should be
communicating
  if(getData == 1) {
    imageReadWriteStore();
  }
}

void imageReadWriteStore() {
```

```

    //locate the camera
    if (cam.begin()) {
        Serial.println("Camera Found:");
    }
    else {
        Serial.println("No camera found?");
        return;
    }

    //set the picture size - you can choose one of 640x480, 320x240 or
160x120
    //cam.setImageSize(VC0706_640x480); //biggest
    //cam.setImageSize(VC0706_320x240); //medium
    cam.setImageSize(VC0706_160x120); //small

    //read the size back from the camera
    uint8_t imgsize = cam.getImageSize();
    Serial.print("Image size: ");
    if (imgsize == VC0706_640x480) Serial.println("640x480");
    if (imgsize == VC0706_320x240) Serial.println("320x240");
    if (imgsize == VC0706_160x120) Serial.println("160x120");

    if (! cam.takePicture()) {
        Serial.println("Failed to snap!");
    }
    else {
        Serial.println("Picture taken!");
    }

    //create an image with the name IMAGExx.JPG
    char filename[13];
    strcpy(filename, "IMAGE00.JPG");
    for (int i = 0; i < 100; i++) {
        filename[5] = '0' + i/10;
        filename[6] = '0' + i%10;
        //create if does not exist, do not open existing, write, sync
after write
        if (! SD.exists(filename)) {
            break;
        }
    }

    //open the file for writing
    File imgFile = SD.open(filename, FILE_WRITE);

    //get the size of the image (frame) taken
    uint16_t jpglen = cam.frameLength();
    Serial.print("Storing ");

```

```

Serial.print(jpglen, DEC);
Serial.print(" byte image.");

pinMode(8, OUTPUT);
//read all the data
byte wCount = 0; //counting # of writes
while (jpglen > 0) {
    uint8_t *buffer;
    uint8_t bytesToRead = min(32, jpglen);
    buffer = cam.readPicture(bytesToRead);
    imgFile.write(buffer, bytesToRead);
    if(++wCount >= 64) { //feedback
        Serial.print('.');
        wCount = 0;
    }
    jpglen -= bytesToRead;
}
imgFile.close();

Serial.println("done!");
}

```


RedBot Code

```
//JEK$DC
//Controls ultrasonic sensors, servo, and hall effect
sensor/LED, no derivative
//Update: 12/17/2014
//XBee serial info: XBeel, 13A20040BAF8AD

//variables for speedometer/hall effect
int hall_pin = 16; //A2 on the redbot
int hall_state = 1;
int revolutions = 0;
float miles_in_inches = 63360;
int mph = 0.0;
float distance = 0.0;
float tire_diameter = 26.2;
float k_ipm_2_mph;
int tire_circumference;
unsigned long last_fall = 0;
unsigned long last_interval = 0;

//variables/libraries for LED
#include <Wire.h> //wire library
#include "Adafruit_LEDBackpack.h" //LED library
#include "Adafruit_GFX.h" //GFX library
Adafruit_AlphaNum4 alpha4 = Adafruit_AlphaNum4();
char displaybuffer[4]; //4 variable array to store speed
String str; //string

//variables for ultrasonic sensor
int anPin = 0; //pin for ultrasonic sensor 1
long ultra1; //additional variables for sensors

//variables for servo
#include <Servo.h> //servo library
Servo servo; //create servo object to control a servo
int servopin = 9; //pin to attach servo to
int pos = 0; // variable to store the servo position
int servodelay = 15; //variable to delay servo
int triggerdistance = 48; //variable to trip servo to rotate
int camangle = 120; //angle that camera will rotate to
int camdelay = 750; //delay for camera

void setup() {
```

```

Serial.begin(9600); //begin serial
Serial.println("");
servo.attach(servopin); //attaches the servo on pin 9 to the
servo object

//begin LED
alpha4.begin(0x70); //pass in the address
alpha4.clear(); //clear alpha at start of program

pinMode(hall_pin, INPUT_PULLUP);

k_ipm_2_mph = 3600000 / miles_in_inches;
tire_circumference = tire_diameter*3.14159;
}

void loop() {
  ultrasonic(); //read trigger distance and find car velocity

  if (ultral < triggerdistance) {
    rotatecam(); //turn camera
    readHallState(); //read hall state to get rpms, speed
    LEDwrite(); //write speed to LED
  }

  Serial.write(0); //keep camera idle
  readHallState(); //read hall state to get rpms, speed
  LEDwrite(); //write speed to LED
}

void ultrasonic() {
  ultral = analogRead(anPin)/2; //convert from reading to
distance

  //print values for troubleshooting
  //Serial.println(ultral);

  delay(50);
}

void rotatecam() {
  for(pos = camangle; pos > 0; pos -= 5) {
    servo.write(pos); // tell servo to go to position in
variable 'pos'

```

```

    delay(servodelay); // waits servodelay for the servo to
reach the position
    }

    delay(camdelay); //wait for camdelay
    Serial.write(1); //send to UNO to take picture
    delay(camdelay); //wait for camdelay

    //return camera to starting position
    for(pos = 0; pos <= camangle; pos += 5) {
        servo.write(pos); // tell servo to go to position in
variable 'pos'
        delay(servodelay); // waits servodelay for the servo to
reach the position
    }
}

void updateSpeedAndDistance() {
    distance = tire_circumference * revolutions;
    distance /= miles_in_inches;
    float ipm = tire_circumference / (float)last_interval;
    mph = ipm * k_ipm_2_mph;
}

void readHallState() {
    int hall_val = digitalRead(hall_pin);

    if (hall_state != hall_val && hall_val == HIGH) {
        revolutions++;
        last_interval = millis()-last_fall;
        last_fall = millis();
    }

    hall_state = hall_val;
    updateSpeedAndDistance();

    //print for troubleshooting
    /*
    Serial.println("");
    Serial.print( hall_state);
    Serial.print("\t");
    Serial.print(mph);
    Serial.print("\t");
    Serial.print(distance);

```

```

    */
}

void LEDwrite() {
    str=String(mph); //converting integer into a string
    str.toCharArray(displaybuffer,4); //string to the character
array
    alpha4.writeDigitAscii(0, displaybuffer[0]);
    alpha4.writeDigitAscii(1, displaybuffer[1]);
    alpha4.writeDigitAscii(2, displaybuffer[2]);
    alpha4.writeDigitAscii(3, displaybuffer[3]);

    alpha4.writeDisplay(); //write to LED display
}

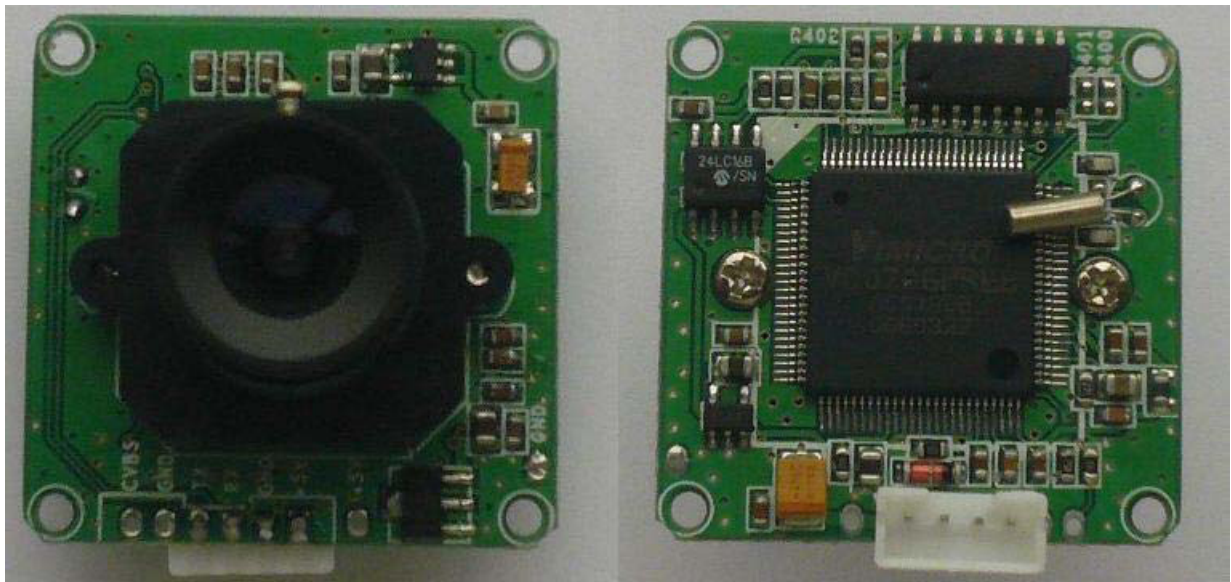
```

PTC08 serial camera module

Manual

Revision 1.02

Sep 20 ,2009

**Putal**

目 录

1	PTC08 PTC08 Serial Camera Introduction.....	3
2	Key performance indicators.....	3
3	camera interface description.....	4
4	Camera protocol.....	7
4.1	Reset Command.....	7
4.2	Camera instruction.....	7
4.3	Reading the film images length instruction.....	7
4.4	Reading the film images data instructions.....	7
4.5	Stop taking commands.....	7
4.6	Instruction Set camera picture compression.....	7
4.7	Set the camera image size command.....	7
4.8	Power down state command.....	7
4.9	Modify the serial port speed command.....	7
4.10	PTC08 camera power initialization process.....	8
5	Field of application.....	8

1 PTC08 串口摄像头简介

PTC08 is the design of Guangzhou Communications Technology Co., Ltd. Thai design and production. It is a camera module that can perform image capture, capture control, data compression, serial transmission used for industrial image capture. It has a built-in high-performance digital signal processing chip that can perform a high proportion of the original image compression. Product image output uses the standard JPEG format, and is easily compatible with various image processing management software. Standard three-wire RS-232 communication interface and a simple image transmission protocol enables the camera can be easily implemented. Optionally, the camera can be fitted with infrared lights and filters so that it can be used in any lighting condition

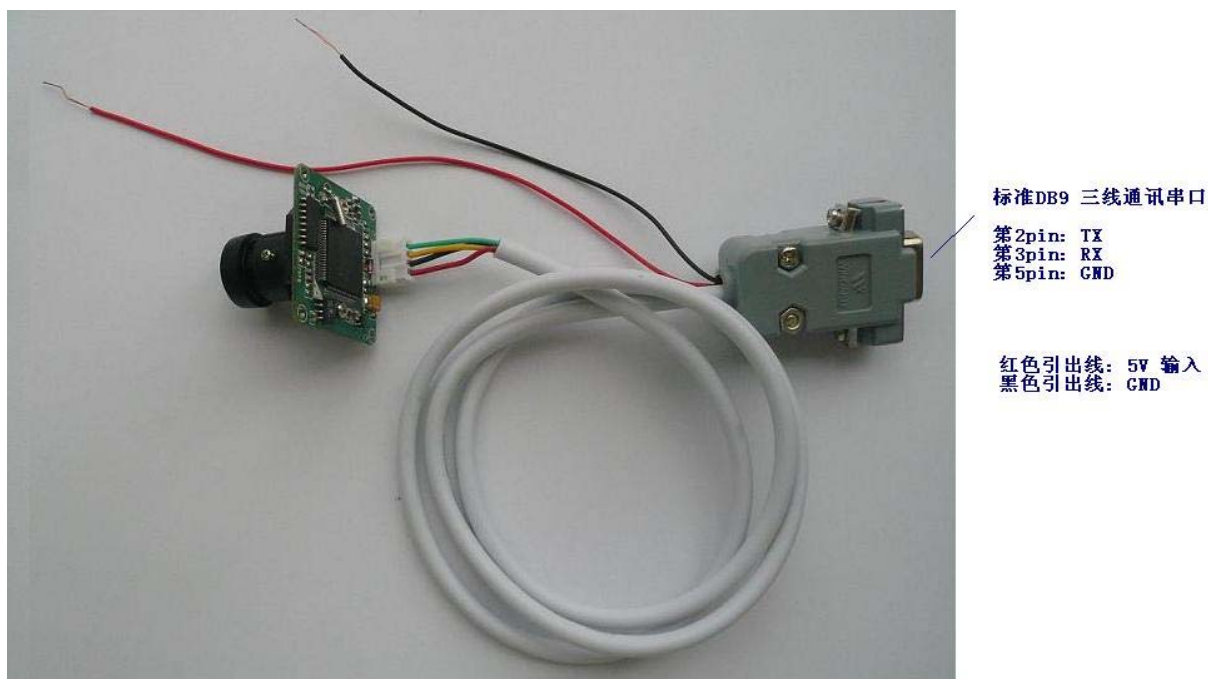
This product's default baud rate is 38400, and other optional Baud rate 9600, 19200, 57600,115200

2. 主要性能指标

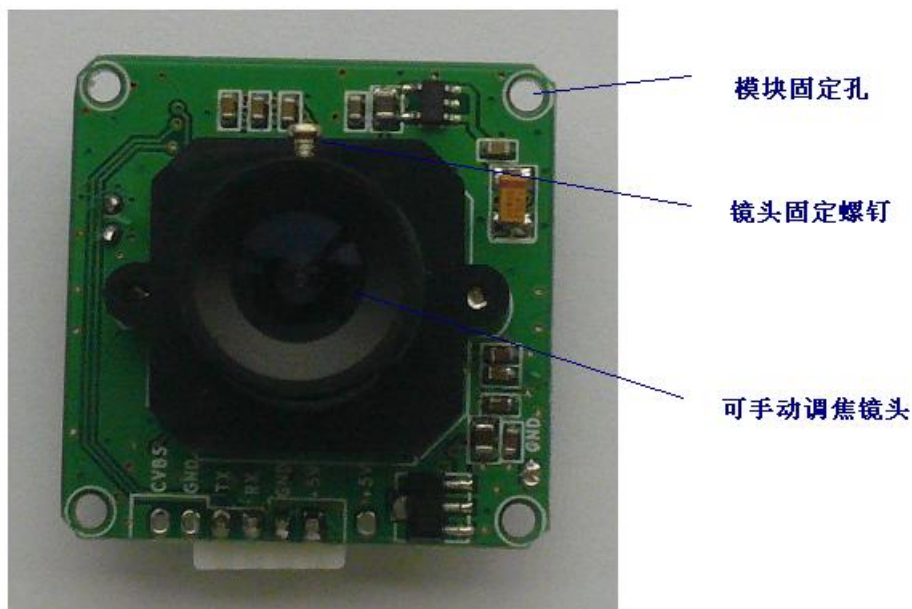
Feature	Parameter
Module size	32mm x 32mm
Image sensor	CMOS 1/4 inch
Megapixels	30 万
Pixel size	5.6um*5.6um
Output format	Standard JPEG/M-JPEG
White balance	Automatic
Exposure	Automatic
Gain	Automatic
Shutter	Electronic rolling shutter
SNR	45DB
Dynamic range	60DB
Max Analog gain	16DB
Frame speed	640*480 30fps
Scan mode	Progressive scan
Viewing angle	120 degrees
Monitoring distance	10 meters, maximum 15meters (adjustable)
Image size	VGA (640*480) , QVGA (320*240), QQVGA (160*120)
Night vision (IR)	Optional
Baud rate	Default 38400 , Maximum 115200
Current draw	75mA
Operating voltage	DC +5V
Communication	RS232 (Three wire TX , RX, GND)

3. 摄像头接口说明

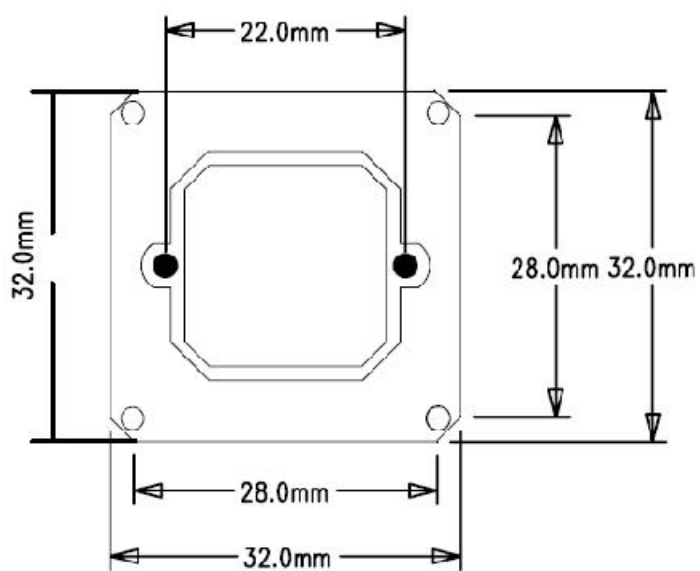
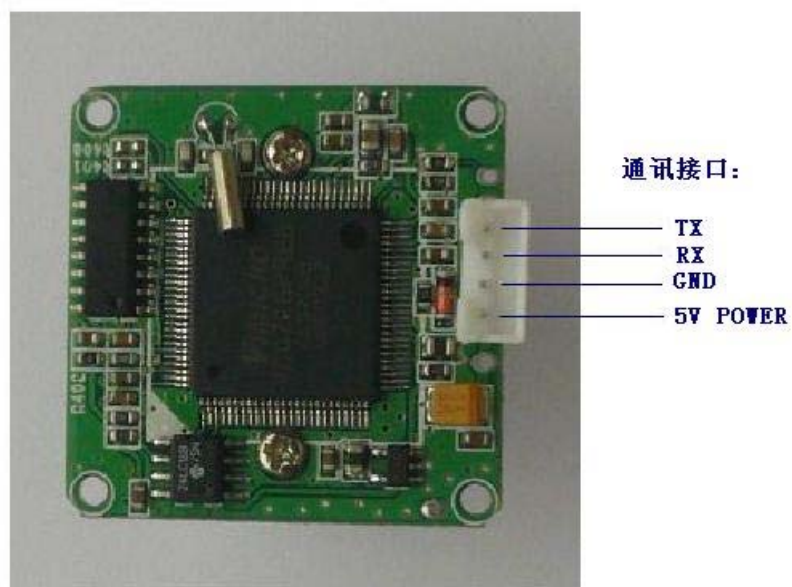
PTC08 串口摄像头模块分为模块本体和串口连接线两部分。两者之间用可任意插拔的4pin 2.0mm 间距的标准插座连接，如下图所示：



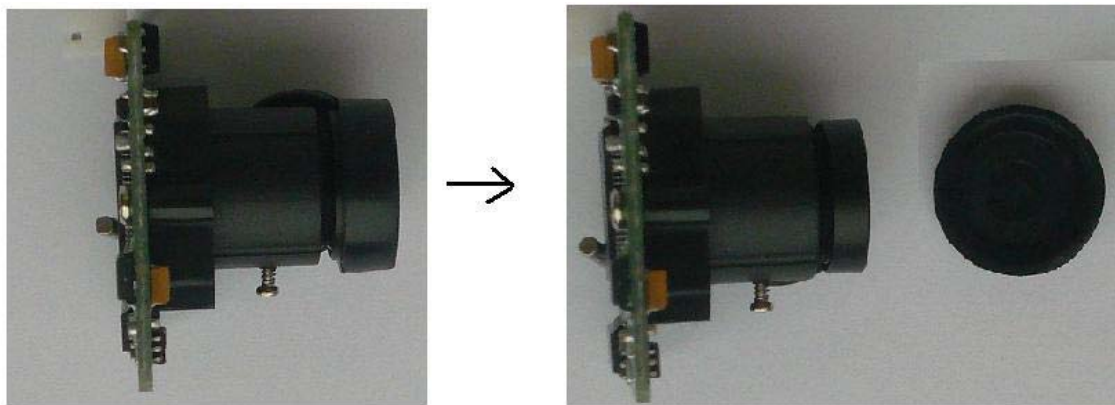
PTC08 模块板+延长线 完整视图



PTC08 TOP面视图

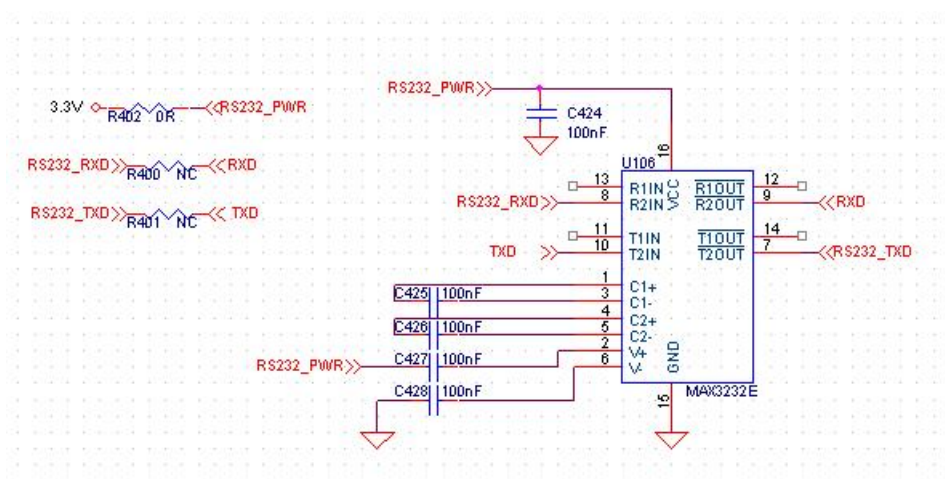


PTC08 模块尺寸图



取镜头盖时，直接往外拔，而不要拧，以免拧动镜头而导致焦距偏离，拍照模糊。

PTC08 摄像头内部默认配有 MAX3232 IC，从 DB9 串口第 2，第 3pin 出来的 TX，RX 是 RS232 电平的信号，可以直接匹配标准 PC 机的串口电平。



此时板上的 3 个选择电阻分别如下（出厂默认配置）：

R400 ---- NC

R401 ---- NC

R401 ---- 0 欧

如果要直接连接单片机或 ARM 等控制器芯片的 TTL 电平串口，则可以去掉 MAX232IC，或者将 R400，R401，R402 改动一下也可以。改动如下：

R400 ---- 0 欧

R401 ---- 0 欧

R402 ---- NC

XBee[®]/XBee-PRO[®] RF Modules

XBee[®]/XBee-PRO[®] RF Modules

RF Module Operation

RF Module Configuration

Appendices



Product Manual v1.xEx - 802.15.4 Protocol

For RF Module Part Numbers: XB24-A...-001, XBP24-A...-001

IEEE[®] 802.15.4 RF Modules by Digi International



Digi International Inc.
11001 Bren Road East
Minnetonka, MN 55343
877 912-3444 or 952 912-3444
<http://www.digi.com>

90000982_B
2009.09.23

© 2009 Digi International, Inc. All rights reserved

The contents of this manual may not be transmitted or reproduced in any form or by any means without the written permission of Digi, Inc.

XBee® and XBee-PRO® are registered trademarks of Digi, Inc.

Technical Support:	Phone:	(866) 765-9885 toll-free U.S.A. & Canada (801) 765-9885 Worldwide 8:00 am - 5:00 pm [U.S. Mountain Time]
	Live Chat:	www.digi.com
	Online Support:	http://www.digi.com/support/eservice/login.jsp
	Email:	rf-experts@digi.com

Contents

1. XBee®/XBee-PRO® RF Modules	4	
Key Features	4	
Worldwide Acceptance	4	
Specifications	5	
Mechanical Drawings	5	
Mounting Considerations	6	
Pin Signals	7	
Electrical Characteristics	8	
2. RF Module Operation	10	
Serial Communications	10	
UART Data Flow	10	
Transparent Operation	11	
API Operation	11	
Flow Control	12	
ADC and Digital I/O Line Support	13	
I/O Data Format	13	
API Support	14	
Sleep Support	14	
DIO Pin Change Detect	14	
Sample Rate (Interval)	14	
I/O Line Passing	15	
Configuration Example	15	
XBee®/XBee-PRO® Networks	16	
Peer-to-Peer	16	
NonBeacon (w/ Coordinator)	16	
Association	17	
XBee®/XBee-PRO® Addressing	20	
Unicast Mode	20	
Broadcast Mode	20	
Modes of Operation	21	
Idle Mode	21	
Transmit/Receive Modes	21	
Sleep Mode	23	
Command Mode	25	
3. RF Module Configuration	26	
Programming the RF Module	26	
Programming Examples	26	
Remote Configuration Commands	27	
Sending a Remote Command	27	
Applying Changes on Remote	27	
Remote Command Responses	27	
Command Reference Tables	27	
Command Descriptions	36	
API Operation	57	
		API Frame Specifications 57
		API Types 58
		Appendix A: Agency Certifications 64
		United States (FCC) 64
		OEM Labeling Requirements 64
		FCC Notices 64
		FCC-Approved Antennas (2.4 GHz) 65
		Approved Antennas 67
		Canada (IC) 68
		Labeling Requirements 68
		Japan 68
		Labeling Requirements 68
		Appendix B: Additional Information 69
		1-Year Warranty 69

1. XBee®/XBee-PRO® RF Modules

The XBee and XBee-PRO RF Modules were engineered to meet IEEE 802.15.4 standards and support the unique needs of low-cost, low-power wireless sensor networks. The modules require minimal power and provide reliable delivery of data between devices.

The modules operate within the ISM 2.4 GHz frequency band and are pin-for-pin compatible with each other.



Key Features

Long Range Data Integrity

XBee

- Indoor/Urban: up to 100' (30 m)
- Outdoor line-of-sight: up to 300' (90 m)
- Transmit Power: 1 mW (0 dBm)
- Receiver Sensitivity: -92 dBm

XBee-PRO

- Indoor/Urban: up to 300' (90 m), 200' (60 m) for International variant
- Outdoor line-of-sight: up to 1 mile (1600 m), 2500' (750 m) for International variant
- Transmit Power: 63mW (18dBm), 10mW (10dBm) for International variant
- Receiver Sensitivity: -100 dBm

RF Data Rate: 250,000 bps

Advanced Networking & Security

Retries and Acknowledgements
DSSS (Direct Sequence Spread Spectrum)
Each direct sequence channels has over 65,000 unique network addresses available
Source/Destination Addressing
Unicast & Broadcast Communications
Point-to-point, point-to-multipoint and peer-to-peer topologies supported

Low Power

XBee

- TX Peak Current: 45 mA (@3.3 V)
- RX Current: 50 mA (@3.3 V)
- Power-down Current: < 10 μ A

XBee-PRO

- TX Peak Current: 250mA (150mA for international variant)
- TX Peak Current (RPSMA module only): 340mA (180mA for international variant)
- RX Current: 55 mA (@3.3 V)
- Power-down Current: < 10 μ A

ADC and I/O line support

Analog-to-digital conversion, Digital I/O
I/O Line Passing

Easy-to-Use

No configuration necessary for out-of box RF communications
Free X-CTU Software (Testing and configuration software)
AT and API Command Modes for configuring module parameters
Extensive command set
Small form factor

Worldwide Acceptance

FCC Approval (USA) Refer to Appendix A [p64] for FCC Requirements.
Systems that contain XBee®/XBee-PRO® RF Modules inherit Digi Certifications.

ISM (Industrial, Scientific & Medical) **2.4 GHz frequency band**

Manufactured under **ISO 9001:2000** registered standards

XBee®/XBee-PRO® RF Modules are optimized for use in the United States, Canada, Australia, Japan, and Europe. Contact Digi for complete list of government agency approvals.



Specifications

Table 1-01. Specifications of the XBee®/XBee-PRO® RF Modules

Specification	XBee	XBee-PRO
Performance		
Indoor/Urban Range	Up to 100 ft (30 m)	Up to 300 ft. (90 m), up to 200 ft (60 m) International variant
Outdoor RF line-of-sight Range	Up to 300 ft (90 m)	Up to 1 mile (1600 m), up to 2500 ft (750 m) international variant
Transmit Power Output (software selectable)	1mW (0 dBm)	63mW (18dBm)* 10mW (10 dBm) for International variant
RF Data Rate	250,000 bps	250,000 bps
Serial Interface Data Rate (software selectable)	1200 bps - 250 kbps (non-standard baud rates also supported)	1200 bps - 250 kbps (non-standard baud rates also supported)
Receiver Sensitivity	-92 dBm (1% packet error rate)	-100 dBm (1% packet error rate)
Power Requirements		
Supply Voltage	2.8 – 3.4 V	2.8 – 3.4 V
Transmit Current (typical)	45mA (@ 3.3 V)	250mA (@3.3 V) (150mA for international variant) RPSMA module only: 340mA (@3.3 V) (180mA for international variant)
Idle / Receive Current (typical)	50mA (@ 3.3 V)	55mA (@ 3.3 V)
Power-down Current	< 10 μ A	< 10 μ A
General		
Operating Frequency	ISM 2.4 GHz	ISM 2.4 GHz
Dimensions	0.960" x 1.087" (2.438cm x 2.761cm)	0.960" x 1.297" (2.438cm x 3.294cm)
Operating Temperature	-40 to 85° C (industrial)	-40 to 85° C (industrial)
Antenna Options	Integrated Whip, Chip or U.FL Connector, RPSMA Connector	Integrated Whip, Chip or U.FL Connector, RPSMA Connector
Networking & Security		
Supported Network Topologies	Point-to-point, Point-to-multipoint & Peer-to-peer	
Number of Channels (software selectable)	16 Direct Sequence Channels	12 Direct Sequence Channels
Addressing Options	PAN ID, Channel and Addresses	PAN ID, Channel and Addresses
Agency Approvals		
United States (FCC Part 15.247)	OUR-XBEE	OUR-XBEEPRO
Industry Canada (IC)	4214A XBEE	4214A XBEEPRO
Europe (CE)	ETSI	ETSI (Max. 10 dBm transmit power output)*
Japan	R201WW07215214	R201WW08215111 (Max. 10 dBm transmit power output)*
Australia	C-Tick	C-Tick

* See Appendix A for region-specific certification requirements.

Antenna Options: The ranges specified are typical when using the integrated Whip (1.5 dBi) and Dipole (2.1 dBi) antennas. The Chip antenna option provides advantages in its form factor; however, it typically yields shorter range than the Whip and Dipole antenna options when transmitting outdoors. For more information, refer to the "XBee Antennas" Knowledgebase Article located on Digi's Support Web site

Mechanical Drawings

Figure 1-01. Mechanical drawings of the XBee®/XBee-PRO® RF Modules (antenna options not shown)

LV-MaxSonar®-EZ™ Series

High Performance Sonar Range Finder

MB1000, MB1010, MB1020, MB1030, MB1040

With 2.5V - 5.5V power the LV-MaxSonar-EZ provides very short to long-range detection and ranging in a very small package. The LV-MaxSonar-EZ detects objects from 0-inches to 254-inches (6.45-meters) and provides sonar range information from 6-inches out to 254-inches with 1-inch resolution. Objects from 0-inches to 6-inches typically range as 6-inches¹. The interface output formats included are pulse width output, analog voltage output, and RS232 serial output. Factory calibration and testing is completed with a flat object. ¹See Close Range Operation



Features

- Continuously variable gain for control and side lobe suppression
- Object detection to zero range objects
- 2.5V to 5.5V supply with 2mA typical current draw
- Readings can occur up to every 50mS, (20-Hz rate)
- Free run operation can continually measure and output range information
- Triggered operation provides the range reading as desired
- Interfaces are active simultaneously
- Serial, 0 to Vcc, 9600 Baud, 81N
- Analog, (Vcc/512) / inch
- Pulse width, (147uS/inch)

- Learns ringdown pattern when commanded to start ranging
- Designed for protected indoor environments
- Sensor operates at 42KHz
- High output square wave sensor drive (double Vcc)

Benefits

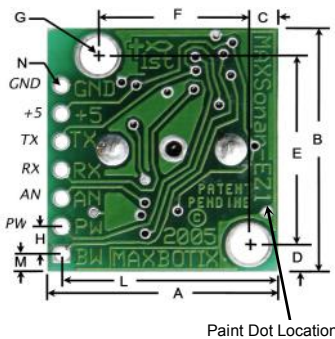
- Very low cost ultrasonic rangefinder
- Reliable and stable range data
- Quality beam characteristics
- Mounting holes provided on the circuit board
- Very low power ranger, excellent for multiple sensor or battery-based systems
- Fast measurement cycles

- Sensor reports the range reading directly and frees up user processor
- Choose one of three sensor outputs
- Triggered externally or internally

Applications and Uses

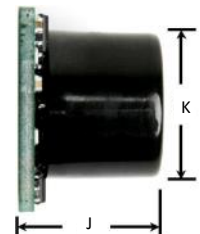
- UAV blimps, micro planes and some helicopters
- Bin level measurement
- Proximity zone detection
- People detection
- Robot ranging sensor
- Autonomous navigation
- Multi-sensor arrays
- Distance measuring
- Long range object detection
- Wide beam sensitivity

LV-MaxSonar-EZ Mechanical Dimensions



A	0.785"	19.9 mm	H	0.100"	2.54 mm
B	0.870"	22.1 mm	J	0.610"	15.5 mm
C	0.100"	2.54 mm	K	0.645"	16.4 mm
D	0.100"	2.54 mm	L	0.735"	18.7 mm
E	0.670"	17.0 mm	M	0.065"	1.7 mm
F	0.510"	12.6 mm	N	0.038" dia.	1.0 mm dia.
G	0.124" dia.	3.1 mm dia.	weight, 4.3 grams		

Part Number	MB1000	MB1010	MB1020	MB1030	MB1040
Paint Dot Color	Black	Brown	Red	Orange	Yellow



Close Range Operation

Applications requiring 100% reading-to-reading reliability should not use MaxSonar sensors at a distance closer than 6 inches. Although most users find MaxSonar sensors to work reliably from 0 to 6 inches for detecting objects in many applications, MaxBotix® Inc. does not guarantee operational reliability for objects closer than the minimum reported distance. Because of ultrasonic physics, these sensors are unable to achieve 100% reliability at close distances.

Warning: Personal Safety Applications

We do not recommend or endorse this product be used as a component in any personal safety applications. This product is not designed, intended or authorized for such use. These sensors and controls do not include the self-checking redundant circuitry needed for such use. Such unauthorized use may create a failure of the MaxBotix® Inc. product which may result in personal injury or death. MaxBotix® Inc. will not be held liable for unauthorized use of this component.

About Ultrasonic Sensors

Our ultrasonic sensors are in air, non-contact object detection and ranging sensors that detect objects within an area. These sensors are not affected by the color or other visual characteristics of the detected object. Ultrasonic sensors use high frequency sound to detect and localize objects in a variety of environments. Ultrasonic sensors measure the time of flight for sound that has been transmitted to and reflected back from nearby objects. Based upon the time of flight, the sensor then outputs a range reading.

Pin Out Description

- Pin 1-BW-** *Leave open or hold low for serial output on the TX output. When BW pin is held high the TX output sends a pulse (instead of serial data), suitable for low noise chaining.
- Pin 2-PW-** This pin outputs a pulse width representation of range. The distance can be calculated using the scale factor of 147uS per inch.
- Pin 3-AN-** Outputs analog voltage with a scaling factor of ($V_{cc}/512$) per inch. A supply of 5V yields ~9.8mV/in. and 3.3V yields ~6.4mV/in. The output is buffered and corresponds to the most recent range data.
- Pin 4-RX-** This pin is internally pulled high. The LV-MaxSonar-EZ will continually measure range and output if RX data is left unconnected or held high. If held low the sensor will stop ranging. Bring high for 20uS or more to command a range reading.
- Pin 5-TX-** When the *BW is open or held low, the TX output delivers asynchronous serial with an RS232 format, except voltages are 0-Vcc. The output is an ASCII capital "R", followed by three ASCII character digits representing the range in inches up to a maximum of 255, followed by a carriage return (ASCII 13). The baud rate is 9600, 8 bits, no parity, with one stop bit. Although the voltage of 0-Vcc is outside the RS232 standard, most RS232 devices have sufficient margin to read 0-Vcc serial data. If standard voltage level RS232 is desired, invert, and connect an RS232 converter such as a MAX232. When BW pin is held high the TX output sends a single pulse, suitable for low noise chaining. (no serial data)
- Pin 6-+5V-** Vcc – Operates on 2.5V - 5.5V. Recommended current capability of 3mA for 5V, and 2mA for 3V.
- Pin 7-GND-** Return for the DC power supply. GND (& Vcc) must be ripple and noise free for best operation.

Range "0" Location

The LV-MaxSonar-EZ reports the range to distant targets starting from the front of the sensor as shown in the diagram below.



Range Zero

The range is measured from the front of the transducer.

In general, the LV-MaxSonar-EZ will report the range to the leading edge of the closest detectable object. Target detection has been characterized in the sensor beam patterns.

Sensor Minimum Distance

The sensor minimum reported distance is 6-inches (15.2 cm). However, the LV-MaxSonar-EZ will range and report targets to the front sensor face. Large targets closer than 6-inches will typically range as 6-inches.

Sensor Operation from 6-inches to 20-inches

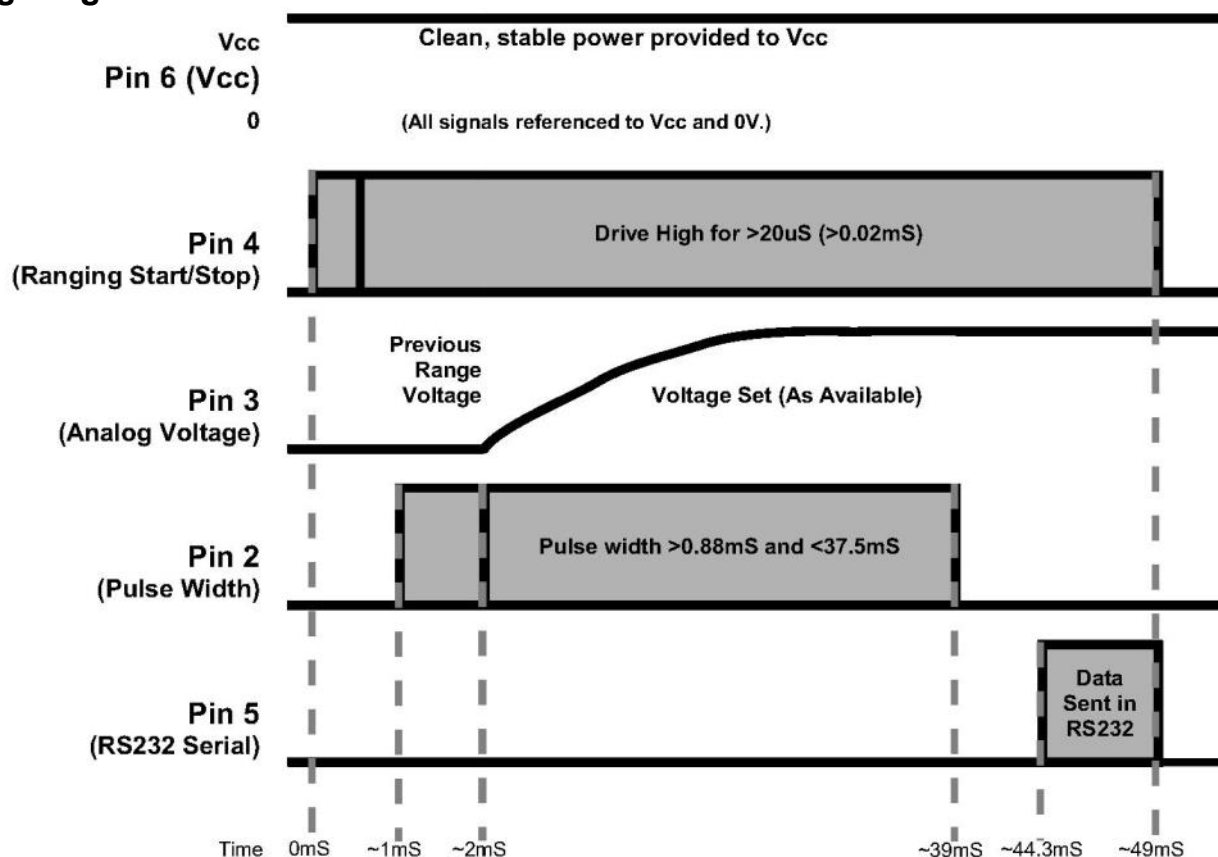
Because of acoustic phase effects in the near field, objects between 6-inches and 20-inches may experience acoustic phase cancellation of the returning waveform resulting in inaccuracies of up to 2-inches. These effects become less prevalent as the target distance increases, and has not been observed past 20-inches.

General Power-Up Instruction

Each time the LV-MaxSonar-EZ is powered up, it will calibrate during its first read cycle. The sensor uses this stored information to range a close object. It is important that objects not be close to the sensor during this calibration cycle. The best sensitivity is obtained when the detection area is clear for fourteen inches, but good results are common when clear for at least seven inches. If an object is too close during the calibration cycle, the sensor may ignore objects at that distance.

The LV-MaxSonar-EZ does not use the calibration data to temperature compensate for range, but instead to compensate for the sensor ringdown pattern. If the temperature, humidity, or applied voltage changes during operation, the sensor may require recalibration to reacquire the ringdown pattern. Unless recalibrated, if the temperature increases, the sensor is more likely to have false close readings. If the temperature decreases, the sensor is more likely to have reduced up close sensitivity. To recalibrate the LV-MaxSonar-EZ, cycle power, then command a read cycle.

Timing Diagram



Timing Description

250mS after power-up, the LV-MaxSonar-EZ is ready to accept the RX command. If the RX pin is left open or held high, the sensor will first run a calibration cycle (49mS), and then it will take a range reading (49mS). After the power up delay, the first reading will take an additional ~100mS. Subsequent readings will take 49mS. The LV-MaxSonar-EZ checks the RX pin at the end of every cycle. Range data can be acquired once every 49mS.

Each 49mS period starts by the RX being high or open, after which the LV-MaxSonar-EZ sends the transmit burst, after which the pulse width pin (PW) is set high. When a target is detected the PW pin is pulled low. The PW pin is high for up to 37.5mS if no target is detected. The remainder of the 49mS time (less 4.7mS) is spent adjusting the analog voltage to the correct level. When a long distance is measured immediately after a short distance reading, the analog voltage may not reach the exact level within one read cycle. During the last 4.7mS, the serial data is sent.

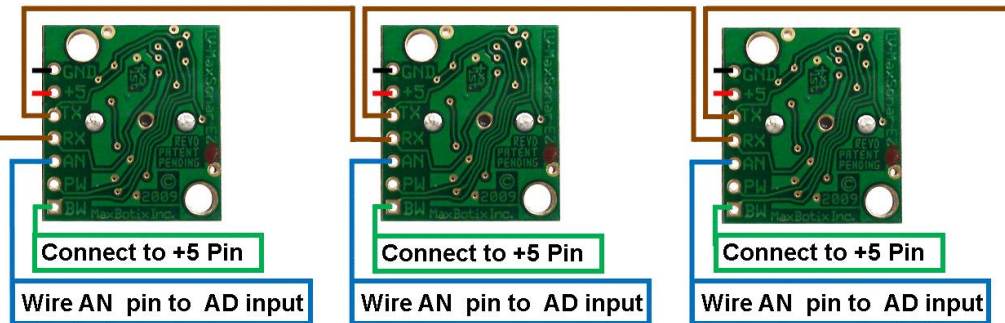
The LV-MaxSonar-EZ timing is factory calibrated to one percent at five volts, and in use is better than two percent. In addition, operation at 3.3V typically causes the objects range, to be reported, one to two percent further than actual.

Using Multiple Sensors in a single system

When using multiple ultrasonic sensors in a single system, there can be interference (cross-talk) from the other sensors. MaxBotix Inc., has engineered and supplied a solution to this problem for the LV-MaxSonar-EZ sensors. The solution is referred to as chaining. We have 3 methods of chaining that work well to avoid the issue of cross-talk.

The first method is AN Output Commanded Loop. The first sensor will range, then trigger the next sensor to range and so on for all the sensor in the array. Once the last sensor has ranged, the array stops until the first sensor is triggered to range again. Below is a diagram on how to set this up.

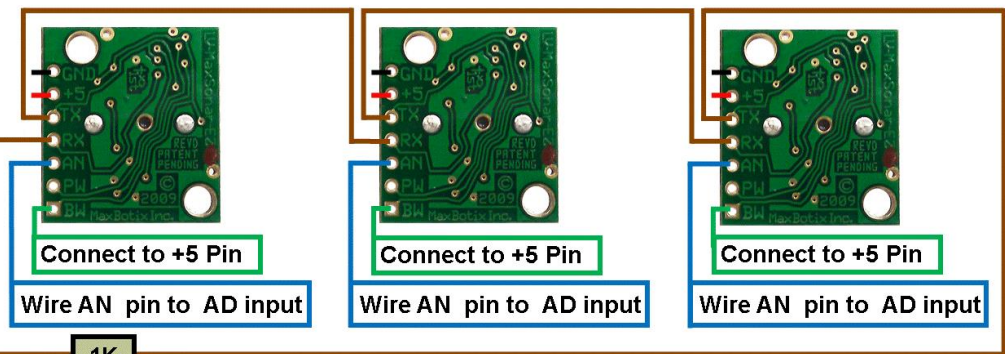
To command a range cycle, bring the RX pin high for a time greater than 20uS but less than 48mS and return to ground. This will start the sensor chain. Repeat this every time you want the sensors to range.



Repeat to add as many sensors as desired

The next method is AN Output Constantly Looping. The first sensor will range, then trigger the next sensor to range and so on for all the sensor in the array. Once the last sensor has ranged, it will trigger the first sensor in the array to range again and will continue this loop indefinitely. Below is a diagram on how to set this up.

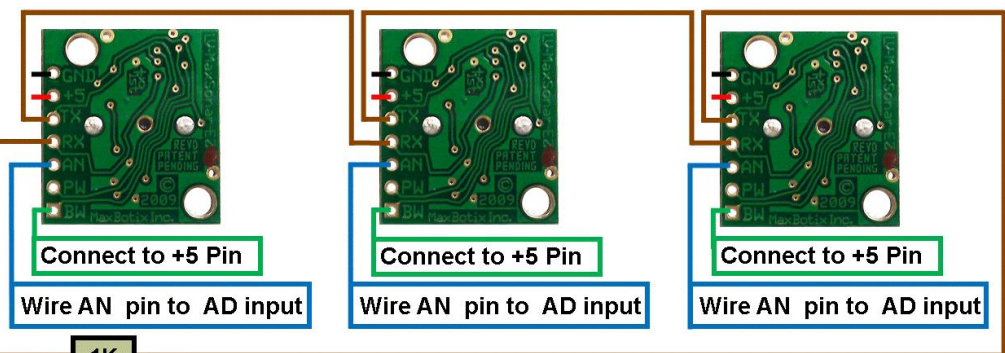
To start the continuous loop, bring the RX pin high for a time greater than 20uS but less than 48mS and return to ground or a high impedance state. This will start the sensor chain. To stop the chain, remove power from the sensors.



Repeat to add as many sensors as desired

The final method is AN Output Simultaneous Operation. This method does not work in all applications and is sensitive to how the other sensors in the array are positioned in comparison to each other. Testing is recommend to verify this method will work for your application. All the sensors RX pins are conned together and triggered at the same time

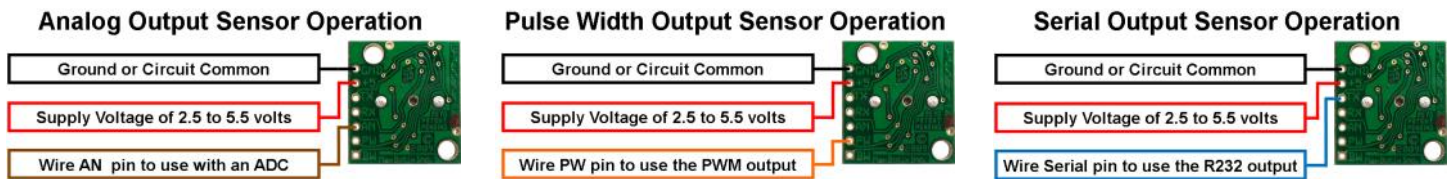
To start the continuous loop, bring the RX pin high for a time greater than 20uS but less than 48mS and return to ground or a high impedance state. This will start the sensor chain. To stop the chain, remove power from the sensors.



Repeat to add as many sensors as desired

Independent Sensor Operation

The LV-MaxSonar-EZ sensors have the capability to operate independently when the user desires. When using the LV-MaxSonar-EZ sensors in single or independent sensor operation, it is easiest to allow the sensor to free-run. Free-run is the default mode of operation for all of the MaxBotix Inc., sensors. The LV-MaxSonar-EZ sensors have three separate outputs that update the range data simultaneously: Analog Voltage, Pulse Width, and RS232 Serial. Below are diagrams on how to connect the sensor for each of the three outputs when operating in a single or independent sensor operating environment.



Selecting an LV-MaxSonar-EZ

Different applications require different sensors. The LV-MaxSonar-EZ product line offers varied sensitivity to allow you to select the best sensor to meet your needs.

The LV-MaxSonar-EZ Sensors At a Glance

People Detection Wide Beam High Sensitivity	Best Balance	Large Targets Narrow Beam Noise Tolerance
MB1000	MB1010	MB1020
		MB1030
		MB1040

The diagram above shows how each product balances sensitivity and noise tolerance. This does not effect the maximum range, pin outputs, or other operations of the sensor. To view how each sensor will function to different sized targets reference the LV-MaxSonar-EZ Beam Patterns.

Background Information Regarding our Beam Patterns

Each LV-MaxSonar-EZ sensor has a calibrated beam pattern. Each sensor is matched to provide the approximate detection pattern shown in this datasheet. This allows end users to select the part number that matches their given sensing application. Each part number has a consistent field of detection so additional units of the same part number will have similar beam patterns. The beam plots are provided to help identify an estimated detection zone for an application based on the acoustic properties of a target versus the plotted beam patterns.

Each beam pattern is a 2D representation of the detection area of the sensor. The beam pattern is actually shaped like a 3D cone (having the same detection pattern both vertically and horizontally). Detection patterns for dowels are used to show the beam pattern of each sensor. Dowels are long cylindered targets of a given diameter. The dowels provide consistent target detection characteristics for a given size target which allows easy comparison of one MaxSonar sensor to another MaxSonar sensor.

People Sensing:

For users that desire to detect people, the detection area to the 1-inch diameter dowel, in general, represents the area that the sensor will reliably detect people.

For each part number, the four patterns (A, B, C, and D) represent the detection zone for a given target size. Each beam pattern shown is determined by the sensor's part number and target size.

The actual beam angle changes over the full range. Use the beam pattern for a specific target at any given distance to calculate the beam angle for that target at the specific distance. Generally, smaller targets are detected over a narrower beam angle and a shorter distance. Larger targets are detected over a wider beam angle and a longer range.

MB1000 LV-MaxSonar-EZ0

The LV-MaxSonar-EZ0 is the highest sensitivity and widest beam sensor of the LV-MaxSonar-EZ sensor series. The wide beam makes this sensor ideal for a variety of applications including people detection, autonomous navigation, and wide beam applications.

MB1000

LV-MaxSonar®-EZ0™ Beam Pattern

Sample results for measured beam pattern are shown on a 30-cm grid. The detection pattern is shown for dowels of varying diameters that are placed in front of the sensor

A 6.1-mm (0.25-inch) diameter dowel

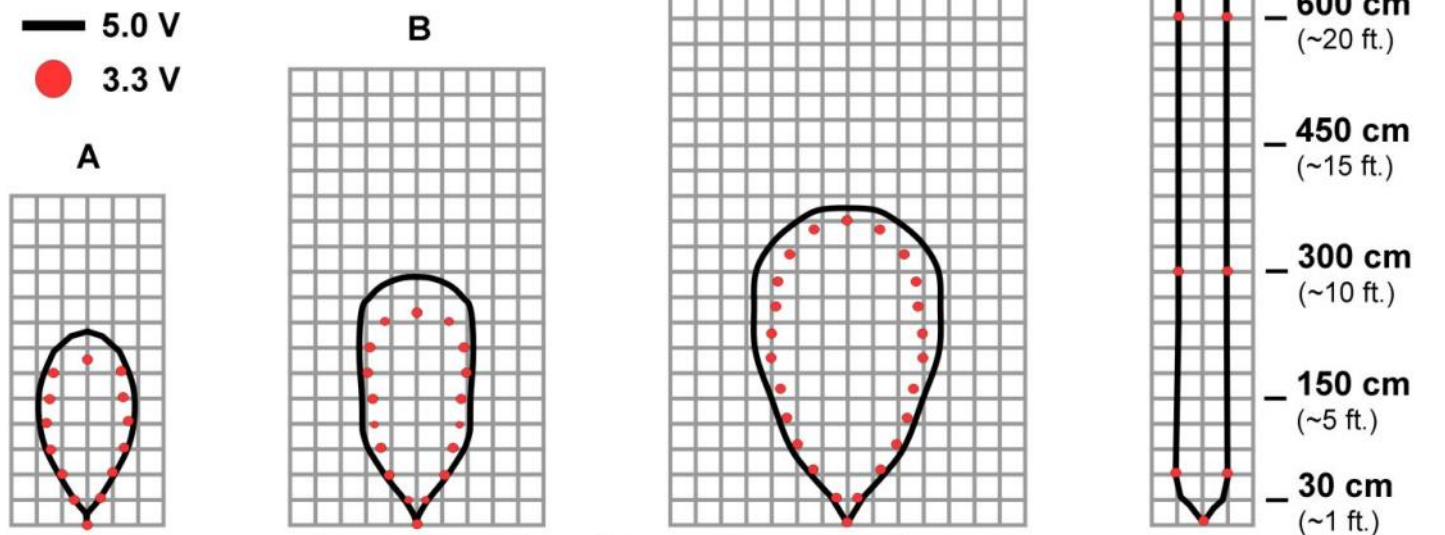
B 2.54-cm (1-inch) diameter dowel

C 8.89-cm (3.5-inch) diameter dowel

D 11-inch wide board moved left to right with the board parallel to the front sensor face.

This shows the sensor's range capability.

Note: For people detection the pattern typically falls between charts A and B.



Beam Characteristics are Approximate

Beam Pattern drawn to a 1:95 scale for easy comparison to our other products.

MB1000 Features and Benefits

- Widest and most sensitive beam pattern in LV-MaxSonar-EZ line
- Low power consumption
- Easy to use interface
- Will pick up the most noise clutter when compared to other sensors in the LV-MaxSonar-EZ line
- Detects smaller objects
- Best sensor to detect soft object in LV-MaxSonar-EZ line
- Requires use of less sensors to cover a given area
- Can be powered by many different types of power sources
- Can detect people up to approximately 10 feet

MB1000 Applications and Uses

- Great for people detection
- Security
- Motion detection
- Used with battery power
- Autonomous navigation
- Educational and hobby robotics
- Collision avoidance

MB1010 LV-MaxSonar-EZ1

The LV-MaxSonar-EZ1 is the original MaxSonar product. This is our most popular indoor ultrasonic sensor and is a great low-cost general-purpose sensor for a customer not sure of which LV-MaxSonar-EZ sensor to use.

MB1010

LV-MaxSonar®-EZ1™ Beam Pattern

Sample results for measured beam pattern are shown on a 30-cm grid. The detection pattern is shown for dowels of varying diameters that are placed in front of the sensor

A 6.1-mm (0.25-inch) diameter dowel

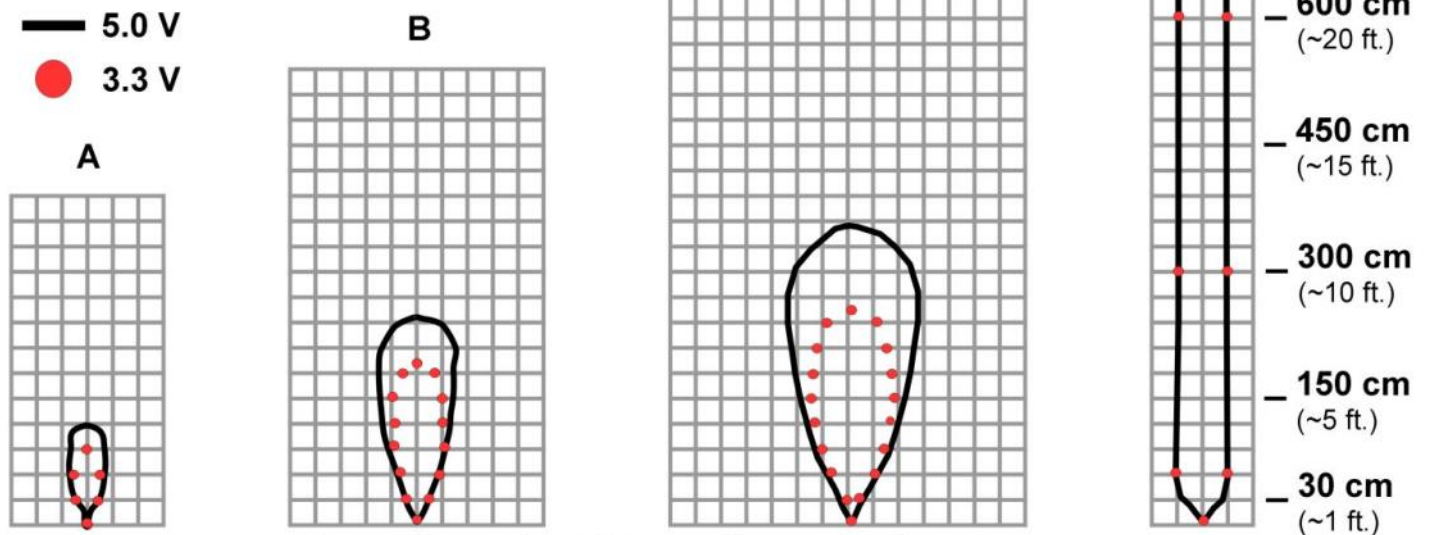
B 2.54-cm (1-inch) diameter dowel

C 8.89-cm (3.5-inch) diameter dowel

D 11-inch wide board moved left to right with the board parallel to the front sensor face.

This shows the sensor's range capability.

Note: For people detection the pattern typically falls between charts A and B.



Beam Characteristics are Approximate

Beam Pattern drawn to a 1:95 scale for easy comparison to our other products.

MB1010 Features and Benefits

- Most popular ultrasonic sensor
- Low power consumption
- Easy to use interface
- Can detect people to 8 feet
- Great balance between sensitivity and object rejection
- Can be powered by many different types of power sources

MB1010 Applications and Uses

- Great for people detection
- Security
- Motion detection
- Used with battery power
- Autonomous navigation
- Educational and hobby robotics
- Collision avoidance

MB1020 LV-MaxSonar-EZ2

The LV-MaxSonar-EZ2 is a good compromise between sensitivity and side object rejection. The LV-MaxSonar-EZ2 is an excellent choice for applications that require slightly less side object detection and sensitivity than the MB1010 LV-MaxSonar-EZ1.

MB1020

LV-MaxSonar®-EZ2™ Beam Pattern

Sample results for measured beam pattern are shown on a 30-cm grid. The detection pattern is shown for dowels of varying diameters that are placed in front of the sensor

A 6.1-mm (0.25-inch) diameter dowel

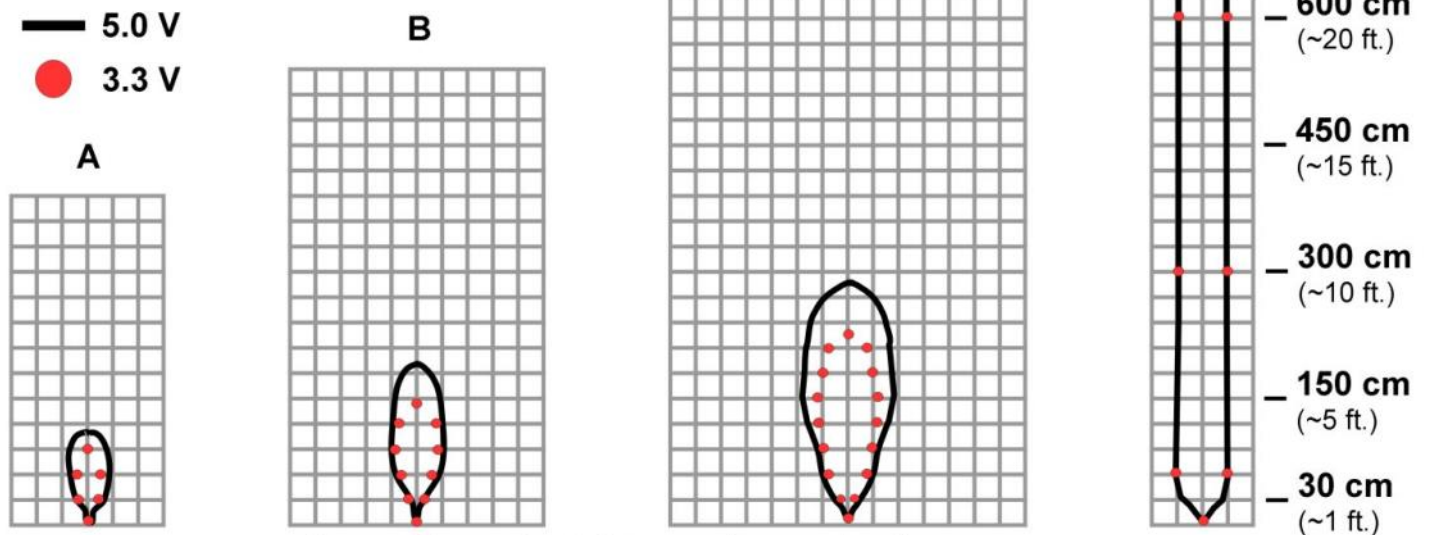
B 2.54-cm (1-inch) diameter dowel

C 8.89-cm (3.5-inch) diameter dowel

D 11-inch wide board moved left to right with the board parallel to the front sensor face.

This shows the sensor's range capability.

Note: For people detection the pattern typically falls between charts A and B.



Beam Characteristics are Approximate

Beam Pattern drawn to a 1:95 scale for easy comparison to our other products.

MB1020 Features and Benefits

- Great for applications where the MB1010 is too sensitive.
- Excellent side object rejection
- Can be powered by many different types of power sources
- Can detect people up to approximately 6 feet

MB1020 Applications and Uses

- Landing flying objects
- Used with battery power
- Autonomous navigation
- Educational and hobby robotics
- Large object detection

MB1030 LV-MaxSonar-EZ3

The LV-MaxSonar-EZ3 is a narrow beam sensor with good side object rejection. The LV-MaxSonar-EZ3 has slightly wider beam width than the MB1040 LV-MaxSonar-EZ4 which makes it a good choice for when the LV-MaxSonar-EZ4 does not have enough sensitivity for the application.

MB1030

LV-MaxSonar®-EZ3™ Beam Pattern

Sample results for measured beam pattern are shown on a 30-cm grid. The detection pattern is shown for dowels of varying diameters that are placed in front of the sensor

A 6.1-mm (0.25-inch) diameter dowel

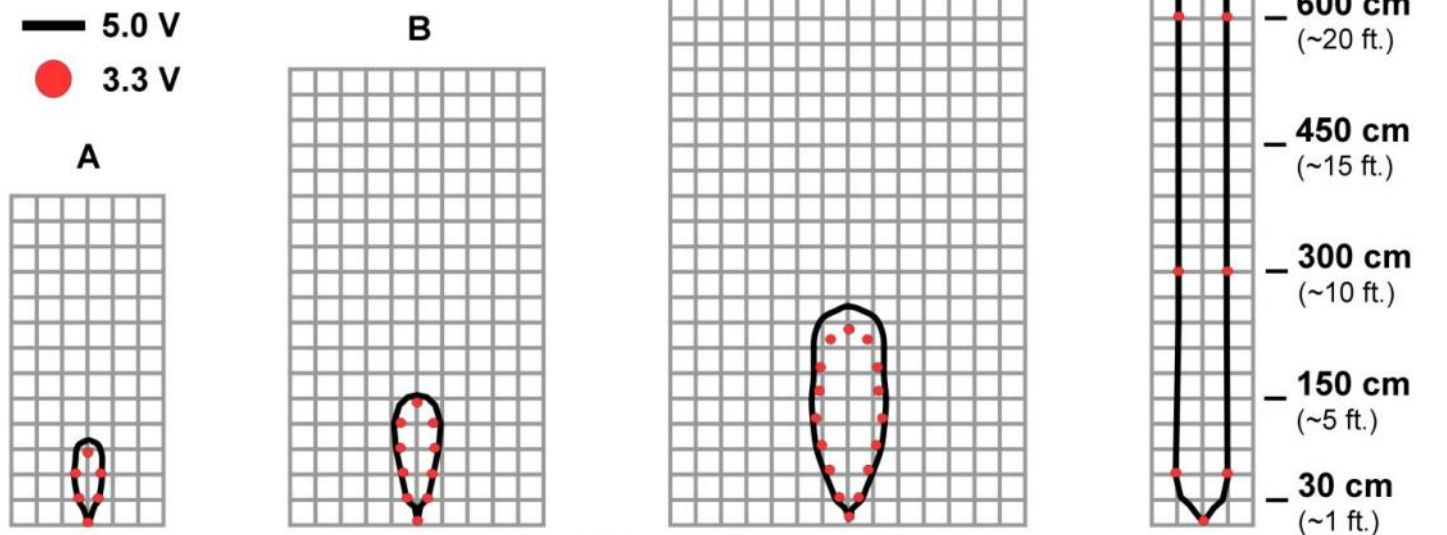
B 2.54-cm (1-inch) diameter dowel

C 8.89-cm (3.5-inch) diameter dowel

D 11-inch wide board moved left to right with the board parallel to the front sensor face.

This shows the sensor's range capability.

Note: For people detection the pattern typically falls between charts A and B.



Beam Characteristics are Approximate

Beam Pattern drawn to a 1:95 scale for easy comparison to our other products.

MB1030 Features and Benefits

- Excellent side object rejection
- Low power consumption
- Easy to use interface
- Great for when MB1040 is not sensitive enough
- Large object detection
- Can be powered by many different types of power sources

- Can detect people up to approximately 5 feet

MB1030 Applications and Uses

- Landing flying objects
- Used with battery power
- Autonomous navigation
- Educational and hobby robotics

MB1040 LV-MaxSonar-EZ4

The LV-MaxSonar-EZ4 is the narrowest beam width sensor that is also the least sensitive to side objects offered in the LV-MaxSonar-EZ sensor line. The LV-MaxSonar-EZ4 is an excellent choice when only larger objects need to be detected.

MB1040

LV-MaxSonar®-EZ4™ Beam Pattern

Sample results for measured beam pattern are shown on a 30-cm grid. The detection pattern is shown for dowels of varying diameters that are placed in front of the sensor

A 6.1-mm (0.25-inch) diameter dowel

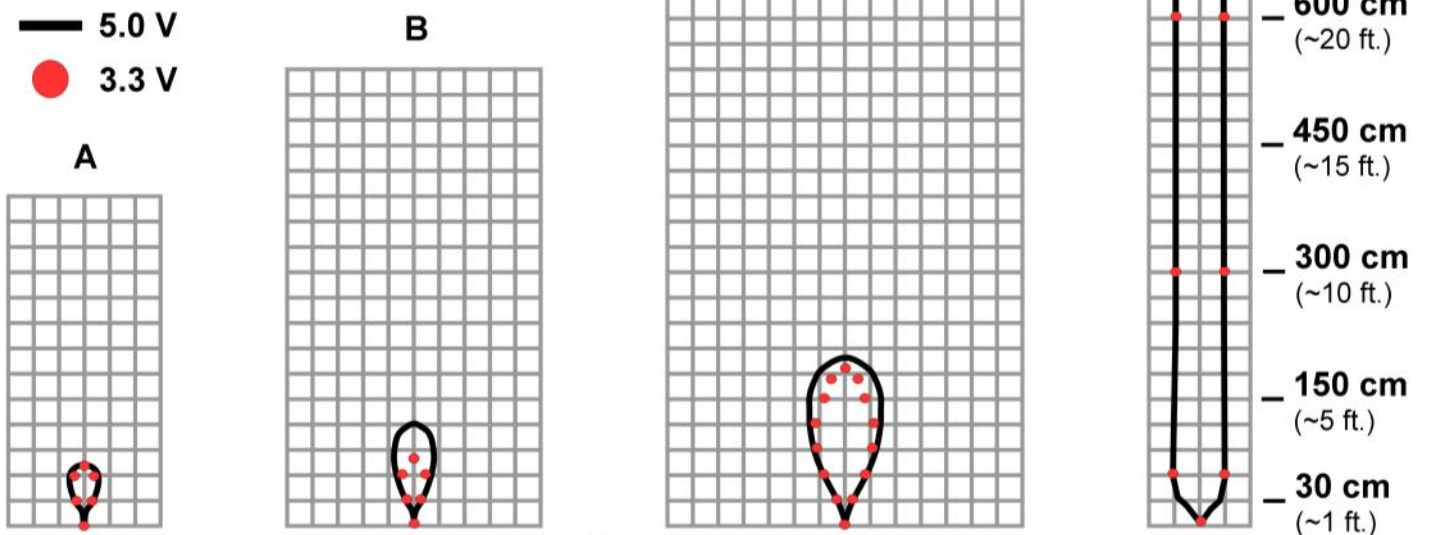
B 2.54-cm (1-inch) diameter dowel

C 8.89-cm (3.5-inch) diameter dowel

D 11-inch wide board moved left to right with the board parallel to the front sensor face.

This shows the sensor's range capability.

Note: For people detection the pattern typically falls between charts A and B.



Beam Characteristics are Approximate

Beam Pattern drawn to a 1:95 scale for easy comparison to our other products.

MB1040 Features and Benefits

- Best side object rejection in the LV-MaxSonar-EZ sensor line
- Low power consumption
- Easy to use interface
- Best for large object detection
- Can be powered by many different types of power sources
- Can detect people up to approximately 4 feet

MB1040 Applications and Uses

- Landing flying objects
- Used with battery power
- Autonomous navigation
- Educational and hobby robotics
- Collision avoidance

Have the right sensor for your application?

Select from this product list for Protected and Non-Protected Environments.

Protected Environments



Accessories — More information is online.

MB7954 — Shielded Cable

The MaxSonar Connection Wire is used to reduce interference caused by electrical noise on the lines. This cable is a great solution to use when running the sensors at a long distance or in an area with a lot of EMI and electrical noise.



MB7950 — XL-MaxSonar-WR Mounting Hardware

The MB7950 Mounting Hardware is selected for use with our outdoor ultrasonic sensors. The mounting hardware includes a steel lock nut and two O-ring (Buna-N and Neoprene) each optimal for different applications.

MB7955 / MB7956 / MB7957 / MB7958 / MB7972 — HR-MaxTemp

The HR-MaxTemp is an optional accessory for the HR-MaxSonar. The HR-MaxTemp connects to the HR-MaxSonar for automatic temperature compensation without self heating.

MB7961 — Power Supply Filter

The power supply filter is recommended for applications with unclean power or electrical noise.

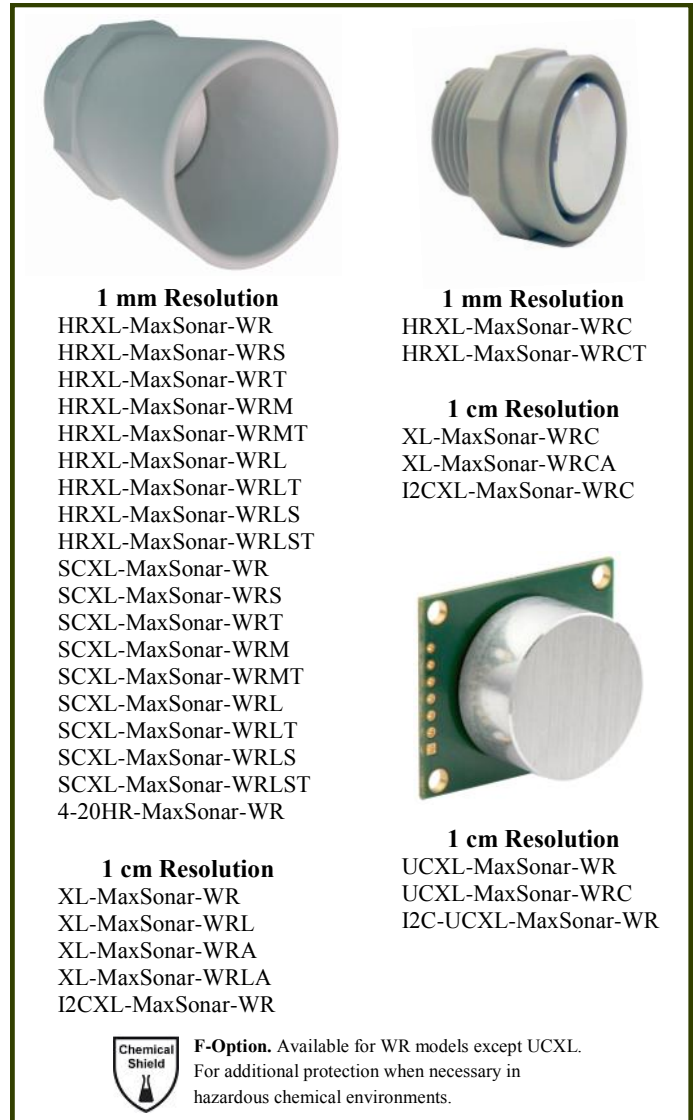
MB7962 / MB7963 / MB7964 / MB7965 — Micro-B USB Connection Cable

The MB7962, MB7963, MB7964 and MB7965 Micro-B USB cables are USB 2.0 compliant and backwards compatible with USB 1.0 standards. Varying lengths.

MB7973 — CE Lightning/Surge Protector

The MB7973 adds protection required to meet the Lightning/Surge IEC61000-4-5 specification.

Non-Protected Environments



Features and Benefits

- ❑ Wide operating voltage range from 3.5V to 24V
- ❑ High magnetic sensitivity – Multi-purpose
- ❑ CMOS technology
- ❑ Chopper-stabilized amplifier stage
- ❑ Low current consumption
- ❑ Open drain output
- ❑ Thin SOT23 3L and flat TO-92 3L both RoHS Compliant packages

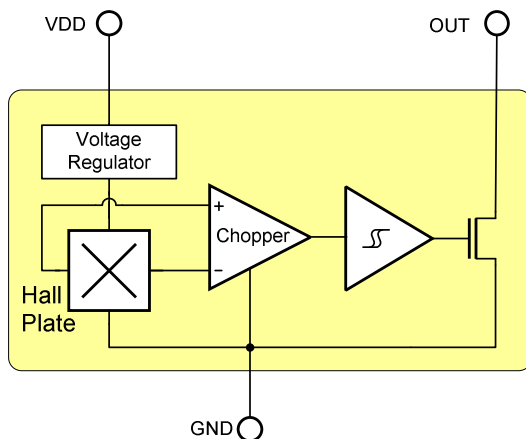
Application Examples

- ❑ Automotive, Consumer and Industrial
- ❑ Solid-state switch
- ❑ Brushless DC motor commutation
- ❑ Speed detection
- ❑ Linear position detection
- ❑ Angular position detection
- ❑ Proximity detection

Ordering Information

Part No.	Temperature Code	Package Code
US1881	E (-40 °C to 85 °C)	SE (TSOT-3L)
US1881	E (-40 °C to 85 °C)	UA (TO-92)
US1881	K (-40 °C to 125 °C)	SE (TSOT-3L)
US1881	K (-40 °C to 125 °C)	UA (TO-92)
US1881	L (-40 °C to 150 °C)	SE (TSOT-3L)
US1881	L (-40 °C to 150 °C)	UA (TO-92)

1 Functional Diagram



2 General Description

The Melexis US1881 is a Hall-effect latch designed in mixed signal CMOS technology.

The device integrates a voltage regulator, Hall sensor with dynamic offset cancellation system, Schmitt trigger and an open-drain output driver, all in a single package.

Thanks to its wide operating voltage range and extended choice of temperature range, it is quite suitable for use in automotive, industrial and consumer applications.

The device is delivered in a Thin Small Outline Transistor (TSOT) for surface mount process and in a Plastic Single In Line (TO-92 flat) for through-hole mount.

Both 3-lead packages are RoHS compliant.

Table of Contents

1 Functional Diagram	1
2 General Description.....	1
3 Glossary of Terms	3
4 Absolute Maximum Ratings.....	3
5 Pin Definitions and Descriptions.....	3
6 General Electrical Specifications	4
7 Magnetic Specifications	4
8 Output Behaviour versus Magnetic Pole	4
9 Detailed General Description.....	5
10 Unique Features.....	5
11 Performance Graphs	6
11.1 Magnetic parameters vs. T_A	6
11.2 Magnetic parameters vs. V_{DD}	6
11.3 V_{DSon} vs. T_A	6
11.4 V_{DSon} vs. V_{DD}	6
11.5 I_{DD} vs. T_A	6
11.6 I_{DD} vs. V_{DD}	6
11.7 I_{OFF} vs. T_A	7
11.8 I_{OFF} vs. V_{DD}	7
12 Test Conditions.....	7
12.1 Supply Current.....	7
12.2 Output Saturation Voltage	7
12.3 Output Leakage Current	7
12.4 Magnetic Thresholds	7
13 Application Information.....	8
13.1 Typical Three-Wire Application Circuit	8
13.2 Two-Wire Circuit	8
13.3 Automotive and Harsh, Noisy Environments Three-Wire Circuit	8
14 Application Comments.....	8
15 Standard information regarding manufacturability of Melexis products with different soldering processes.....	9
16 ESD Precautions	9
17 Package Information.....	10
17.1 SE Package (TSOT-3L).....	10
17.2 UA Package (TO-92 flat)	11
18 Disclaimer.....	12

3 Glossary of Terms

MilliTesla (mT), Gauss	Units of magnetic flux density: 1mT = 10 Gauss
RoHS	Restriction of Hazardous Substances
TSOT	Thin Small Outline Transistor (TSOT package) – also referred with the Melexis package code “SE”
ESD	Electro-Static Discharge
BLDC	Brush-Less Direct-Current
Operating Point (B_{OP})	Magnetic flux density applied on the branded side of the package which turns the output driver ON ($V_{OUT} = V_{DSon}$)
Release Point (B_{RP})	Magnetic flux density applied on the branded side of the package which turns the output driver OFF ($V_{OUT} = \text{high}$)

4 Absolute Maximum Ratings

Parameter	Symbol	Value	Units
Supply Voltage	V_{DD}	28	V
Supply Current	I_{DD}	50	mA
Output Voltage	V_{OUT}	28	V
Output Current	I_{OUT}	50	mA
Storage Temperature Range	T_S	-50 to 150	°C
Maximum Junction Temperature	T_J	165	°C

Table 1: Absolute maximum ratings

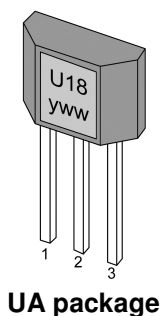
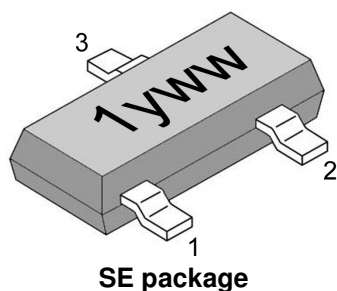
Exceeding the absolute maximum ratings may cause permanent damage. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

Operating Temperature Range	Symbol	Value	Units
Temperature Suffix “E”	T_A	-40 to 85	°C
Temperature Suffix “K”	T_A	-40 to 125	°C
Temperature Suffix “L”	T_A	-40 to 150	°C

5 Pin Definitions and Descriptions

SE Pin №	UA Pin №	Name	Type	Function
1	1	VDD	Supply	Supply Voltage pin
2	3	OUT	Output	Open Drain Output pin
3	2	GND	Ground	Ground pin

Table 2: Pin definitions and descriptions



6 General Electrical Specifications

DC Operating Parameters $T_A = 25^\circ\text{C}$, $V_{DD} = 3.5\text{V}$ to 24V (unless otherwise specified)

Parameter	Symbol	Test Conditions	Min	Typ	Max	Units
Supply Voltage	V_{DD}	Operating	3.5		24	V
Supply Current	I_{DD}	$B < B_{RP}$			5	mA
Output Saturation Voltage	V_{DSon}	$I_{OUT} = 20\text{mA}$, $B > B_{OP}$			0.5	V
Output Leakage Current	I_{OFF}	$B < B_{RP}$, $V_{OUT} = 24\text{V}$		0.3	10	μA
Output Rise Time	t_r	$R_L = 1\text{k}\Omega$, $C_L = 20\text{pF}$		0.25		μs
Output Fall Time	t_f	$R_L = 1\text{k}\Omega$, $C_L = 20\text{pF}$		0.25		μs
Maximum Switching Frequency	F_{SW}			10		KHz
Package Thermal Resistance	R_{TH}	Single layer (1S) Jedec board		301		$^\circ\text{C/W}$

Table 3: Electrical specifications

7 Magnetic Specifications

DC Operating Parameters $V_{DD} = 3.5\text{V}$ to 24V (unless otherwise specified)

Parameter	Symbol	Test Conditions	Min	Typ	Max	Units
Operating Point	B_{OP}	E spec., $T_A = 85^\circ\text{C}$	0.5		9.5	mT
Release Point	B_{RP}		-9.5		-0.5	mT
Hysteresis	B_{HYST}		7		12	mT
Operating Point	B_{OP}	K spec., $T_A = 125^\circ\text{C}$	0.5		9.5	mT
Release Point	B_{RP}		-9.5		-0.5	mT
Hysteresis	B_{HYST}		7		12	mT
Operating Point	B_{OP}	L spec., $T_A = 150^\circ\text{C}$	0.5		9.5	mT
Release Point	B_{RP}		-9.5		-0.5	mT
Hysteresis	B_{HYST}		6		12.5	mT

Table 4: Magnetic specifications

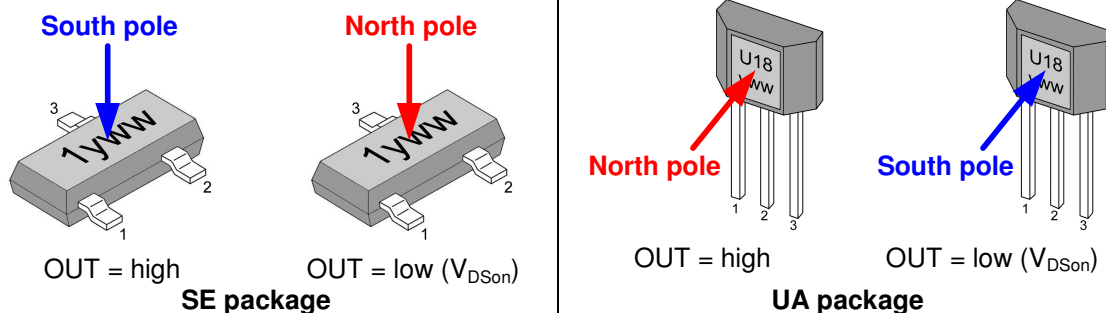
Note 1: For typical values, please refer to the performance graphs in section 11

8 Output Behaviour versus Magnetic Pole

DC Operating Parameters $T_A = -40^\circ\text{C}$ to 150°C , $V_{DD} = 3.5\text{V}$ to 24V (unless otherwise specified)

Parameter	Test Conditions (SE)	OUT (SE)	Test Conditions (UA)	OUT (UA)
South pole	$B < B_{RP}$	High	$B > B_{OP}$	Low
North pole	$B > B_{OP}$	Low	$B < B_{RP}$	High

Table 5: Output behaviour versus magnetic pole



9 Detailed General Description

Based on mixed signal CMOS technology, Melexis US1881 is a Hall-effect device with high magnetic sensitivity. This multi-purpose latch suits most of the application requirements.

The chopper-stabilized amplifier uses switched capacitor technique to suppress the offset generally observed with Hall sensors and amplifiers. The CMOS technology makes this advanced technique possible and contributes to smaller chip size and lower current consumption than bipolar technology. The small chip size is also an important factor to minimize the effect of physical stress.

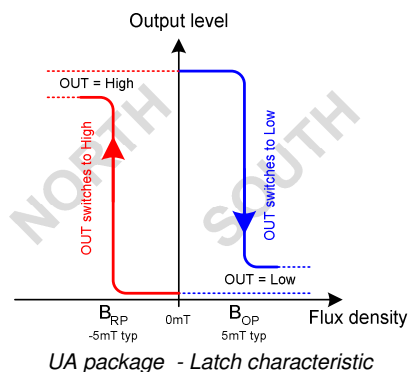
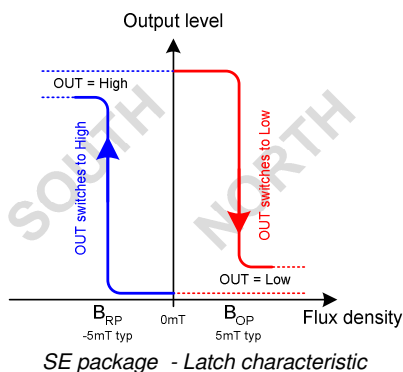
This combination results in more stable magnetic characteristics and enables faster and more precise design.

The wide operating voltage from 3.5V to 24V, low current consumption and large choice of operating temperature range according to “L”, “K” and “E” specification make this device suitable for automotive, industrial and consumer applications.

The output signal is open-drain type. Such output allows simple connectivity with TTL or CMOS logic by using a pull-up resistor tied between a pull-up voltage and the device output.

10 Unique Features

The US1881 exhibits latch magnetic switching characteristics. Therefore, it requires both south and north poles to operate properly.



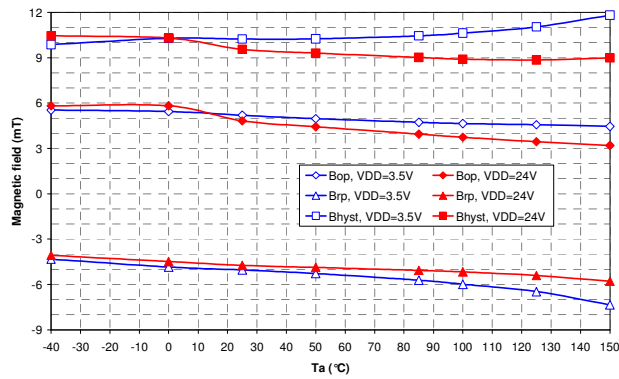
The device behaves as a latch with symmetric operating and release switching points ($B_{OP} = |B_{RP}|$). This means magnetic fields with equivalent strength and opposite direction drive the output high and low.

Removing the magnetic field ($B \rightarrow 0$) keeps the output in its previous state. This latching property defines the device as a magnetic memory.

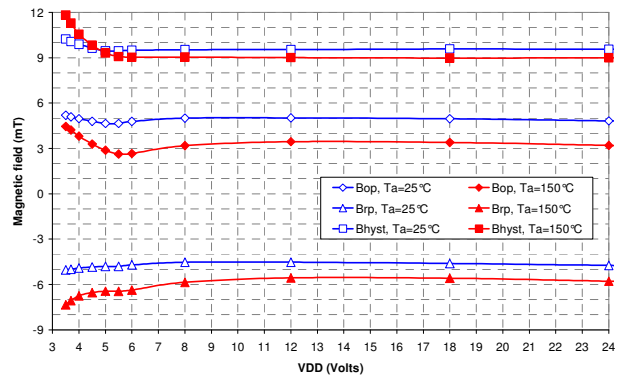
A magnetic hysteresis B_{HYST} keeps B_{OP} and B_{RP} separated by a minimal value. This hysteresis prevents output oscillation near the switching point.

11 Performance Graphs

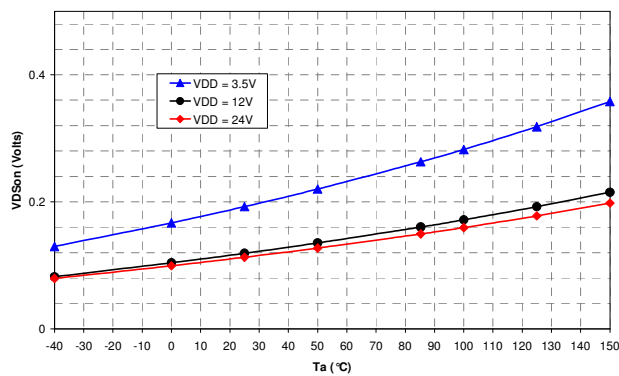
11.1 Magnetic parameters vs. T_A



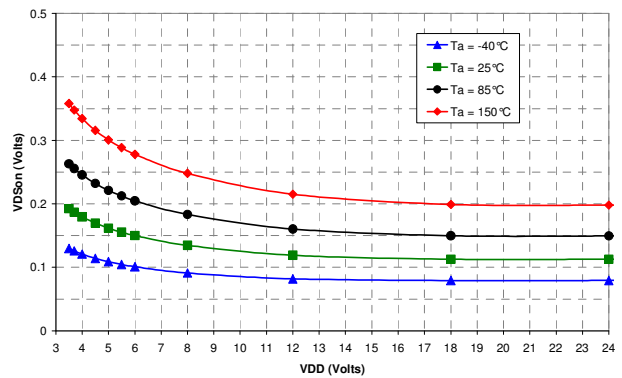
11.2 Magnetic parameters vs. V_{DD}



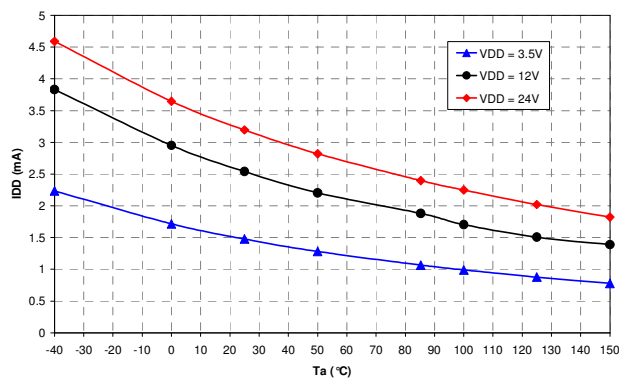
11.3 V_{DSon} vs. T_A



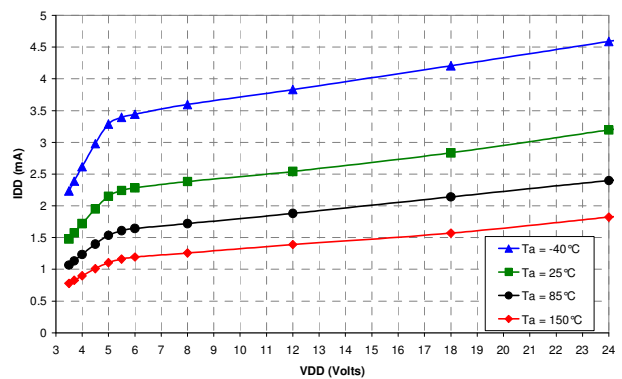
11.4 V_{DSon} vs. V_{DD}



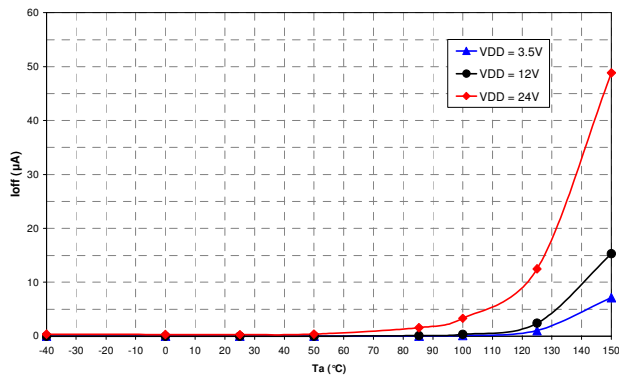
11.5 I_{DD} vs. T_A



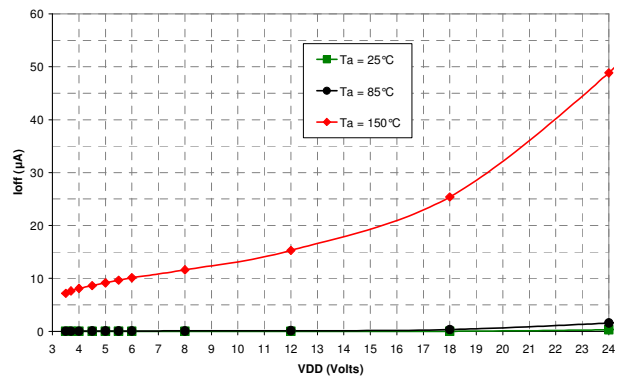
11.6 I_{DD} vs. V_{DD}



11.7 I_{OFF} vs. T_A



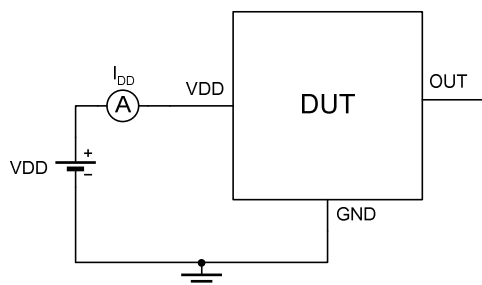
11.8 I_{OFF} vs. V_{DD}



12 Test Conditions

Note : DUT = Device Under Test

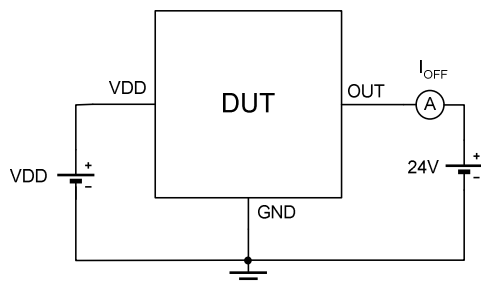
12.1 Supply Current



Note 1 - The supply current I_{DD} represents the static supply current. OUT is left open during measurement.

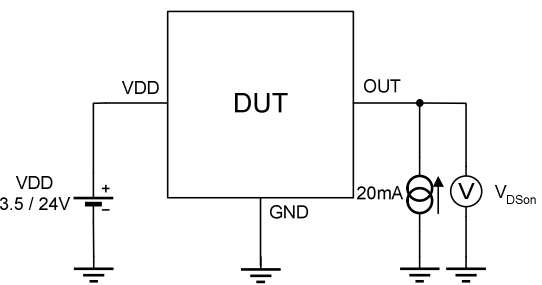
Note 2 - The device is put under magnetic field with $B < B_{RP}$.

12.3 Output Leakage Current



Note 1 - The device is put under magnetic field with $B < B_{RP}$.

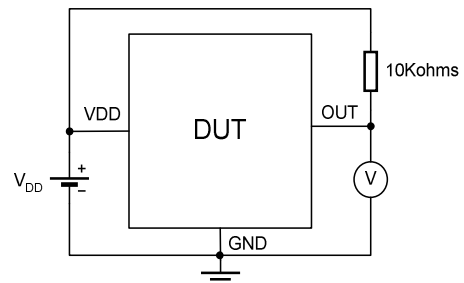
12.2 Output Saturation Voltage



Note 1 - The output saturation voltage $V_{DS(on)}$ is measured at $V_{DD} = 3.5V$ and $V_{DD} = 24V$.

Note 2 - The device is put under magnetic field with $B > B_{OP}$.

12.4 Magnetic Thresholds

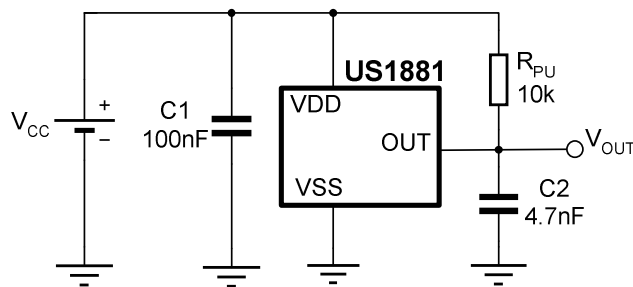


Note 1 - B_{OP} is determined by putting the device under magnetic field swept from B_{RPmin} up to B_{OPmax} until the output is switched on.

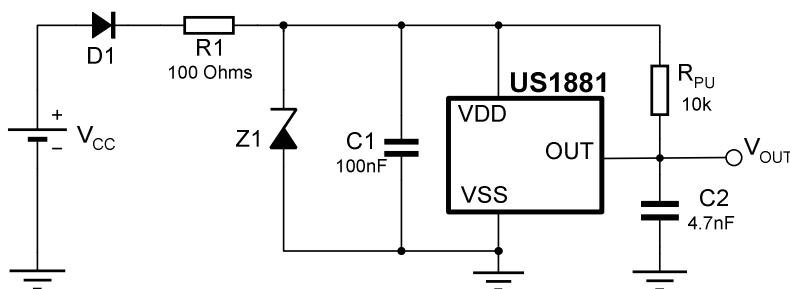
Note 2 - B_{RP} is determined by putting the device under magnetic field swept from B_{OPmax} down to B_{RPmin} until the output is switched off.

13 Application Information

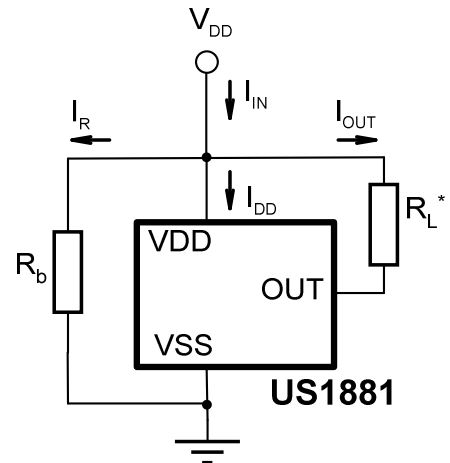
13.1 Typical Three-Wire Application Circuit



13.3 Automotive and Harsh, Noisy Environments Three-Wire Circuit



13.2 Two-Wire Circuit



Note:

With this circuit, precise ON and OFF currents can be detected using only two connecting wires.

The resistors R_L and R_b can be used to bias the input current. Refer to the part specifications for limiting values.

$$B_{RP} : I_{OFF} = I_R + I_{DD} = V_{DD}/R_b + I_{DD}$$

$$B_{OP} : I_{ON} = I_{OFF} + I_{OUT} = I_{OFF} + V_{DD}/R_L$$

14 Application Comments

For proper operation, a 100nF bypass capacitor should be placed as close as possible to the device between the V_{DD} and ground pin.

For reverse voltage protection, it is recommended to connect a resistor or a diode in series with the V_{DD} pin. When using a resistor, three points are important:

- the resistor has to limit the reverse current to 50mA maximum ($V_{CC} / R_1 \leq 50mA$)
- the resulting device supply voltage V_{DD} has to be higher than $V_{DD\ min}$ ($V_{DD} = V_{CC} - R_1 \cdot I_{DD}$)
- the resistor has to withstand the power dissipated in reverse voltage condition ($P_D = V_{CC}^2 / R_1$)

When using a diode, a reverse current cannot flow and the voltage drop is almost constant ($\approx 0.7V$). Therefore, a 100Ω/0.25W resistor for 5V application and a diode for higher supply voltage are recommended. Both solutions provide the required reverse voltage protection.

When a weak power supply is used or when the device is intended to be used in noisy environment, it is recommended that figure 13.3 from the Application Information section is used. The low-pass filter formed by R_1 and C_1 and the zener diode Z_1 bypass the disturbances or voltage spikes occurring on the device supply voltage V_{DD} . The diode D_1 provides additional reverse voltage protection.

15 Standard information regarding manufacturability of Melexis products with different soldering processes

Our products are classified and qualified regarding soldering technology, solderability and moisture sensitivity level according to following test methods:

Reflow Soldering SMD's (Surface Mount Devices)

- IPC/JEDEC J-STD-020
Moisture/Reflow Sensitivity Classification for Nonhermetic Solid State Surface Mount Devices (classification reflow profiles according to table 5-2)
- EIA/JEDEC JESD22-A113
Preconditioning of Nonhermetic Surface Mount Devices Prior to Reliability Testing (reflow profiles according to table 2)

Wave Soldering SMD's (Surface Mount Devices) and THD's (Through Hole Devices)

- EN60749-20
Resistance of plastic- encapsulated SMD's to combined effect of moisture and soldering heat
- EIA/JEDEC JESD22-B106 and EN60749-15
Resistance to soldering temperature for through-hole mounted devices

Iron Soldering THD's (Through Hole Devices)

- EN60749-15
Resistance to soldering temperature for through-hole mounted devices

Solderability SMD's (Surface Mount Devices) and THD's (Through Hole Devices)

- EIA/JEDEC JESD22-B102 and EN60749-21
Solderability

For all soldering technologies deviating from above mentioned standard conditions (regarding peak temperature, temperature gradient, temperature profile etc) additional classification and qualification tests have to be agreed upon with Melexis.

The application of Wave Soldering for SMD's is allowed only after consulting Melexis regarding assurance of adhesive strength between device and board.

Melexis is contributing to global environmental conservation by promoting **lead free** solutions. For more information on qualifications of **RoHS** compliant products (RoHS = European directive on the Restriction Of the use of certain Hazardous Substances) please visit the quality page on our website:

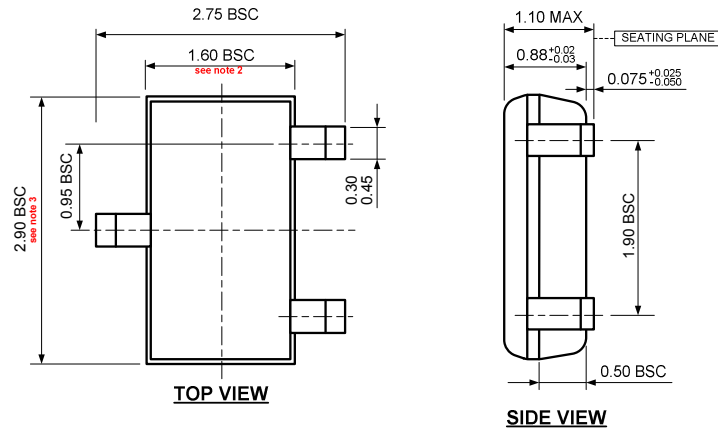
<http://www.melexis.com/quality.asp>

16 ESD Precautions

Electronic semiconductor products are sensitive to Electro Static Discharge (ESD). Always observe Electro Static Discharge control procedures whenever handling semiconductor products.

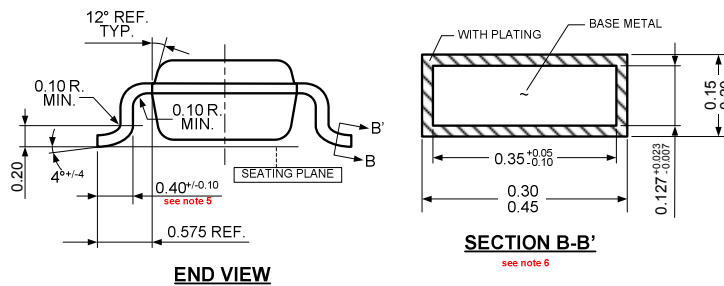
17 Package Information

17.1 SE Package (TSOT-3L)



Notes:

1. All dimensions are in millimeters
2. Outermost plastic extreme width does not include mold flash or protrusions. Mold flash and protrusions shall not exceed 0.15mm per side.
3. Outermost plastic extreme length does not include mold flash or protrusions. Mold flash and protrusions shall not exceed 0.25mm per side.
4. The lead width dimension does not include dambar protrusion. Allowable dambar protrusion shall be 0.07mm total in excess of the lead width dimension at maximum material condition.
5. Dimension is the length of terminal for soldering to a substrate.
6. Dimension on SECTION B-B' are apply to the flat section of the lead between 0.08mm and 0.15mm from the lead tip.
7. Formed lead shall be planar with respect to one another with 0.076mm at seating plane.

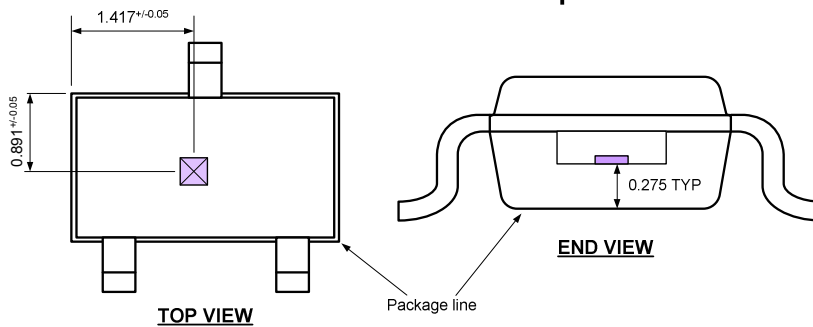


Marking:

Top side : 1yww

1 = part number (US1881)
y = last digit of year
ww = calendar week

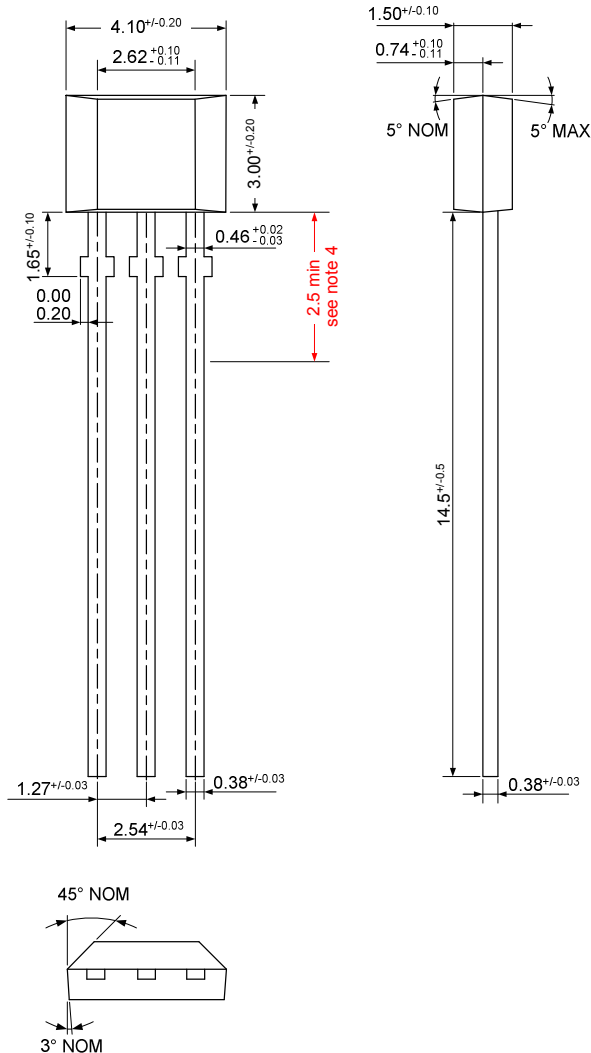
Hall plate location



Notes:

1. All dimensions are in millimeters

17.2 UA Package (TO-92 flat)



Notes:

1. All dimensions are in millimeters
2. Package dimension exclusive molding flash.
3. The end flash shall not exceed 0.127 mm on each side.
4. To preserve reliability, it is recommended to have total lead length equal to 2.5mm minimum, measured from the package line.

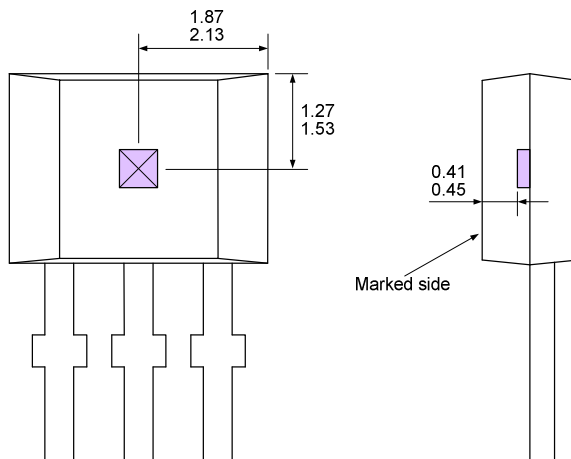
Marking:

1st Line : U18 - Part number (US1881)

2nd Line : yww

y - last digit of year
ww - calendar week

Hall plate location



Notes:

1. All dimensions are in millimeters

18 Disclaimer

Devices sold by Melexis are covered by the warranty and patent indemnification provisions appearing in its Term of Sale. Melexis makes no warranty, express, statutory, implied, or by description regarding the information set forth herein or regarding the freedom of the described devices from patent infringement. Melexis reserves the right to change specifications and prices at any time and without notice. Therefore, prior to designing this product into a system, it is necessary to check with Melexis for current information. This product is intended for use in normal commercial applications. Applications requiring extended temperature range, unusual environmental requirements, or high reliability applications, such as military, medical life-support or life-sustaining equipment are specifically not recommended without additional processing by Melexis for each application.

The information furnished by Melexis is believed to be correct and accurate. However, Melexis shall not be liable to recipient or any third party for any damages, including but not limited to personal injury, property damage, loss of profits, loss of use, interrupt of business or indirect, special incidental or consequential damages, of any kind, in connection with or arising out of the furnishing, performance or use of the technical data herein. No obligation or liability to recipient or any third party shall arise or flow out of Melexis' rendering of technical or other services.

© 2005 Melexis NV. All rights reserved.

For the latest version of this document, go to our website at
www.melexis.com

Or for additional information contact Melexis Direct:

Europe, Africa, Asia:
Phone: +32 1367 0495
E-mail: sales_europe@melexis.com

America:
Phone: +1 603 223 2362
E-mail: sales_usa@melexis.com

ISO/TS 16949 and ISO14001 Certified

0.54" Dual Digit Alphanumeric Displays
Technical Data Sheet

Model No: KWA-541XVB

Features:

- 0.54" (inch) digit height.
- Excellent segment uniformity.
- Sold state reliability.
- Industrial standard size.
- Low power consumption.
- The product itself will remain within RoHS compliant Version.

Descriptions:

The KWA-541XXX series is a larger 13.60mm (0.54") high seven segments display designed for viewing distances up to 7 meters.

These displays provide excellent reliability in bright ambient light.

These devices are made with white segments and black surface.

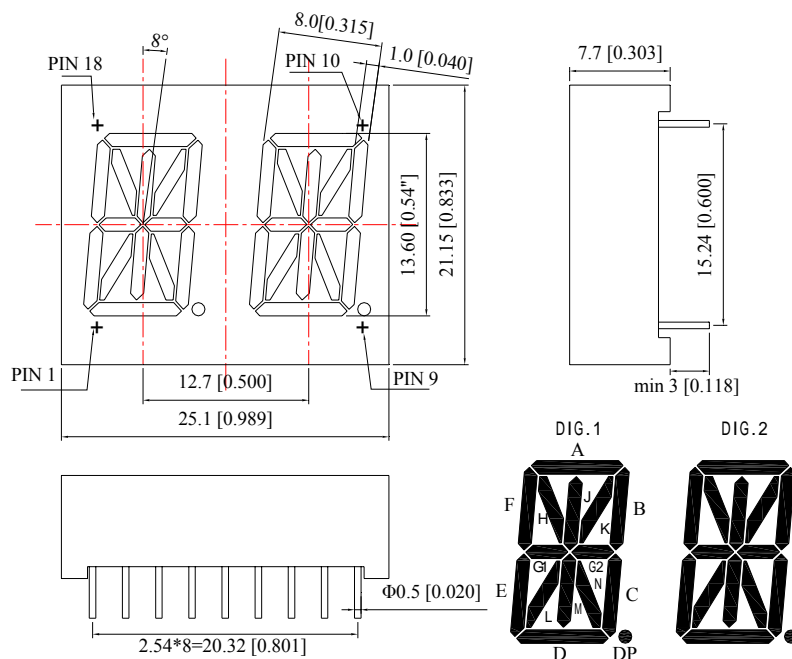
Applications:

- Audio equipment.
- Instrument panels.
- Digital read out display.

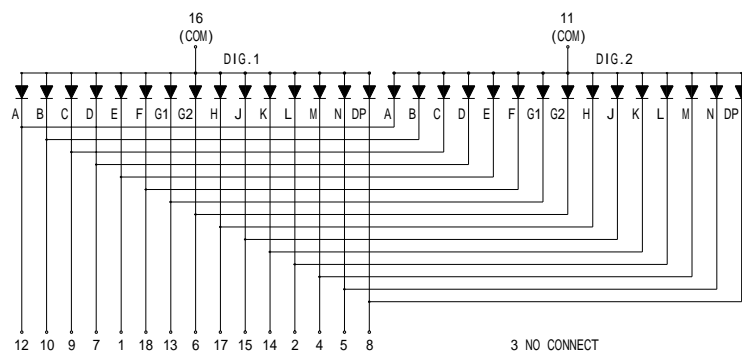
Device Selection Guide:

Model No.	Chip Material	Source Color	Description
KWA-541AVB	AlGaInP	Ultra Red	Common Anode
KWA-541CVB		Ultra Red	Common Cathode

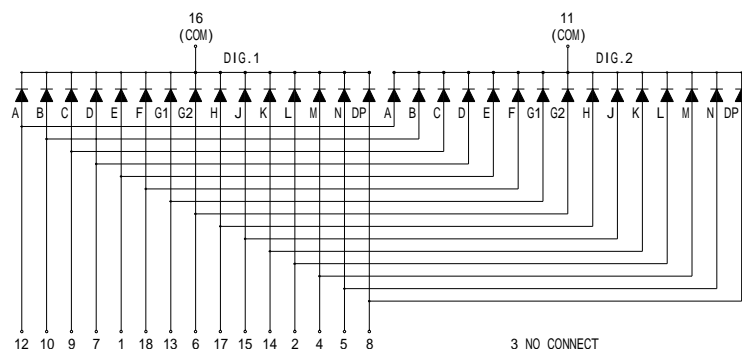
Package Dimension:



KWA-541AVB



KWA-541CVB



Notes:

1. All dimensions are in millimeters (inches).
2. Tolerance is ± 0.25 mm (.010") unless otherwise noted.
3. Specifications are subject to change without notice.

Absolute Maximum Ratings at Ta=25

Parameters	Symbol	Max.	Unit
Power Dissipation (Per Segment)	PD	65	mW
Peak Forward Current (Per Segment) (1/10 Duty Cycle, 0.1ms Pulse Width)	IFP	100	mA
Forward Current (Per Segment)	IF	25	mA
Dating Linear From 25		0.4	mA/
Reverse Voltage	VR	5	V
Operating Temperature Range	Topr	-40 to +80	
Storage Temperature Range	Tstg	-40 to +85	
Soldering Temperature	Tsld	260 for 5 Seconds	

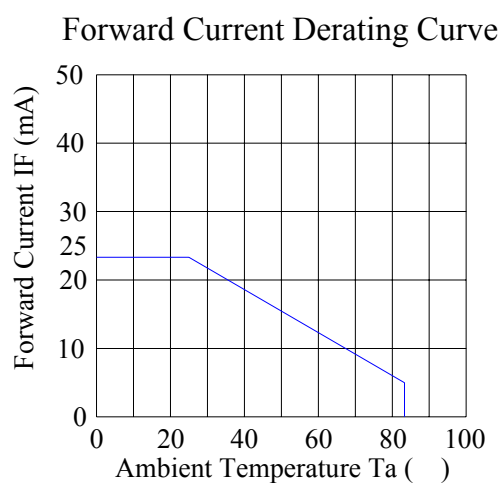
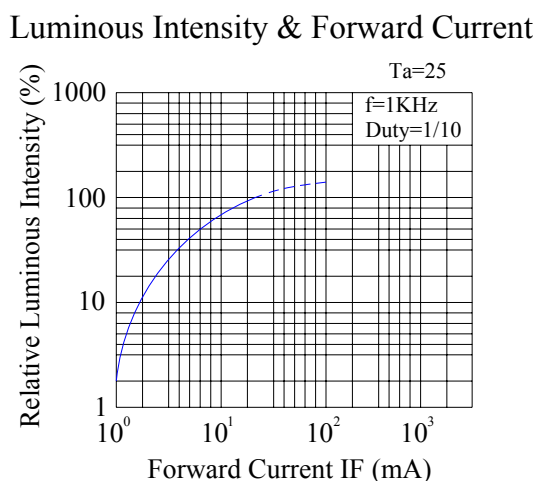
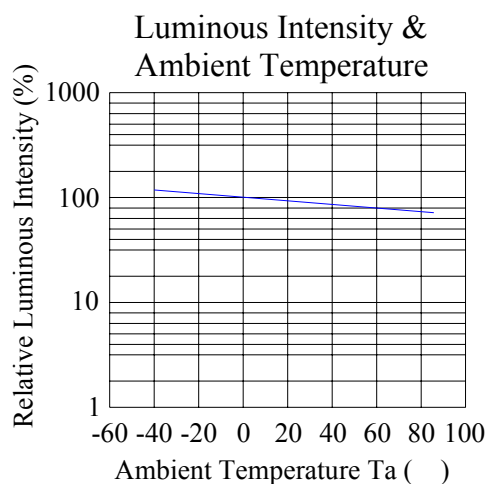
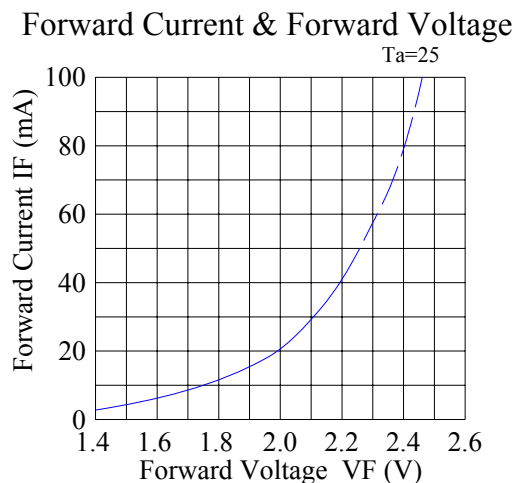
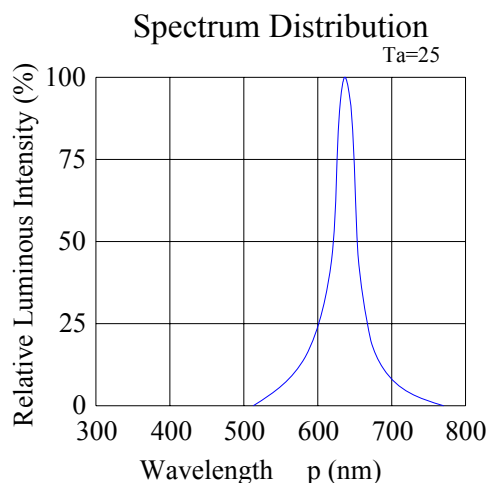
Electrical Optical Characteristics at Ta=25

Parameters	Symbol	Min.	Typ.	Max.	Unit	Test Condition
Luminous Intensity	IV	5.0	10.0	---	mcd	IF=20mA (Note 1)
Luminous Intensity Matching Ratio (Segment To Segment)	I _{v-m}	---	---	2:1		IF=10mA
Peak Emission Wavelength	λ_p	---	632	---	nm	IF=20mA
Dominant Wavelength	λ_d	---	624	---	nm	IF=20mA (Note 2)
Spectral Line Half-Width	λ	---	20	---	nm	IF=20mA
Forward Voltage	VF	---	2.0	2.6	V	IF=20mA
Reverse Current	IR	---	---	50	μ A	VR=5V

Notes:

1. Luminous intensity is measured with a light sensor and filter combination that approximates the CIE eye-response curve.
2. The dominant wavelength (λ_d) is derived from the CIE chromaticity diagram and represents the single wavelength which defines the color of the device.

Typical Electrical / Optical Characteristics Curves (25 Ambient Temperature Unless Otherwise Noted)



Please read the following notes before using the datasheets:

1. Over-current-proof

Customer must apply resistors for protection, otherwise slight voltage shift will cause big current change (Burn out will happen).

2. Storage

2.1 If the package contains a moisture proof bag inside, please don't open the package before using.

2.2 Before opening the package, the LEDs should be kept at 30 °C or less and 80%RH or less.

2.3 The LEDs should be used within a year.

2.4 After opening the package, the LEDs should be kept at 30 °C or less and 60%RH or less.

3. Soldering Iron

Each terminal is to go to the tip of soldering iron temperature less than 260 °C for 5 seconds within once in less than the soldering iron capacity 25W. Leave two seconds and more intervals, and do soldering of each terminal. Be careful because the damage of the product is often started at the time of the hand solder.

4. Soldering

When soldering, for Lamp without stopper type and must be leave a minimum of 3mm clearance from the base of the lens to the soldering point.

To avoided the Epoxy climb up on lead frame and was impact to non-soldering problem, dipping the lens into the solder must be avoided.

Do not apply any external stress to the lead frame during soldering while the LED is at high temperature.

Recommended soldering conditions:

Soldering Iron		Wave Soldering	
Temperature	300 °C Max.	Pre-heat	100 °C Max.
Soldering Time	3 sec. Max. (one time only)	Pre-heat Time	60 sec. Max.
		Solder Wave	260 °C Max.
		Soldering Time	5 sec. Max.

Note: Excessive soldering temperature and / or time might result in deformation of the LED lens or catastrophic failure of the LED.

5. Repairing

Repair should not be done after the LEDs have been soldered. When repairing is unavoidable, a double-head soldering iron should be used (as below figure). It should be confirmed beforehand whether the characteristics of the LEDs will or will not be damaged by repairing.

Mechatronic Jack-O'-Lantern

ME 445

FALL 2014

Matthew Doult

Kevin Masterton



Acknowledgements

We want to thank Dr. Sommer and Mike Robinson for their guidance during ME 445. We learned a lot of mechanical, electrical, and computer engineering concepts that will be invaluable in our careers.

Contents

Introduction.....	3
Concept Development.....	3
Electrical Design.....	3
Adafruit NeoPixel Shield.....	4
Sparkfun Redbot Mainboard.....	5
Adafruit “Music Maker” MP3 Shield.....	6
PIR Sensor	7
Arms & Jaw.....	7
Schematic.....	8
Mechanical Design.....	8
Bill of Materials.....	9
Testing.....	9
Issues and Lessons Learned.....	10
Conclusion.....	10
Appendix A: Arduino Code.....	11
Appendix B: Data Sheets & Schematics.....	18

Introduction

The goal of our project was to develop an interactive mechatronic Halloween decoration. We were drawn to the idea of a robotic jack-o'-lantern candy bowl that would be activated upon sensing motion. It was important to us that the decoration incorporates movement, sound, and lighting effects. The scope of our project included constructing a foam Jack-o'-Lantern body, designing attachments for all components, and programming all components to operate synchronously.

Concept Development

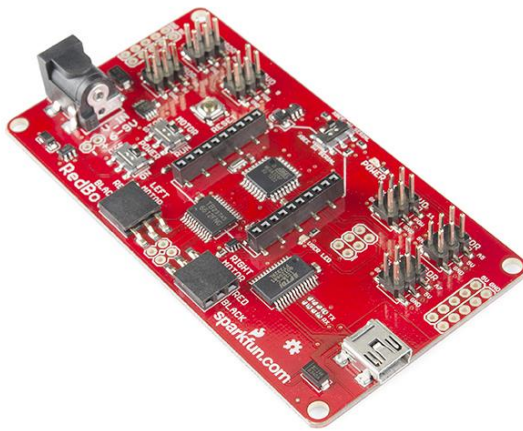
The concept for our Jack-o'-lantern was inspired by the mechatronic characters present throughout Disney World. We wanted to bring the fluid synchronous motion of these robots to a household Halloween decoration. We have both played with the motion activated candy bowls purchased for Halloween, and we wanted to adopt a similar idea.

After brainstorming, we created a list of features that our Halloween decoration would possess. One feature was to express emotion through eye patterns or eye movement. We planned LED matrices for pumpkin's eyes in order to express different emotions and the 40 LED Adafruit Neopixel shield proved to be ideal. The motion activation feature was achieved through the use of a PIR sensor inside the candy bowl of the pumpkin. The sound effects that the decoration would play are accomplished by the use of a MP3 Music Maker shield. Lastly, we decided the movement of the arms and jaw of the Jack-o'-lantern could be created by servos.

Electrical Design

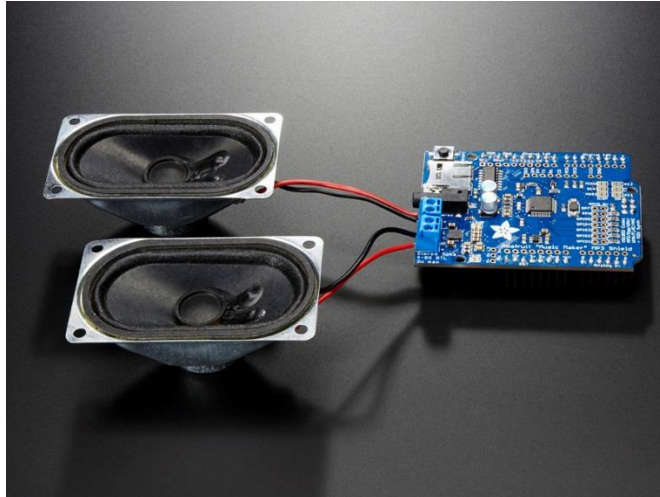
Programming for the animatronic pumpkin involved integrating the four main systems (eyes, motion sensor, arms, and sound) together and making it so they worked in unison. To do this each system was made into its own function, to make the code as simple and easy to follow as possible. The main void loop function is where all of the actions take place. When the motion sensor reads high, the pumpkin will generate a random number between one and four, and based on this number will play a certain combination of eye pattern, movement, and sound. Once the set is complete the pumpkin will wait for more movement and activate again. If a specified amount of time has passed since the last action was completed (twenty seconds currently), the pumpkin will enter a "sleep mode" where it will display a sleeping eye pattern and play a snoring sound. The pumpkin will still search for movement in the candy bowl and will trigger the patterns again when motion is detected.

SparkFun RedBot Mainboard



The SparkFun RedBot Mainboard was chosen to be the main Arduino board for the final design. The RedBot board has several advantages and disadvantages over the Arduino Uno. One of the main issues faced early on in the project was power for the various systems. Each individual electrical component in the design required a +5V supply as well as access to ground. The Uno has problems with this as there is only one +5V output pin and three ground pins. The limited access makes wiring difficult in the already cramped environment of the craft pumpkin. The RedBot provides a solution to this issue by having dedicated +5V and ground headers for each pin, making wiring much simpler and cleaner. The board does come with two downsides though. First is the lower amount of pins available when compared to the Uno. The Uno has thirteen digital I/O pins available, as well as six analog pins which can also be used in digital I/O mode. The RedBot on the other hand has only four digital pins and six analog pins that can be used in digital I/O mode. This creates an issue for the pumpkin due to the amount of outputs needed. The second issue is the lack of shield support, which when coupled with the lack of pins, makes it difficult to use the MP3 shield with the RedBot. The issues with the RedBot were overlooked in favor of the simplicity of wiring, but changes eventually had to be made to accommodate for the issues in other systems.

Adafruit "Music Maker" MP3 Shield for Arduino w/3W Stereo Amp - v1.0



Playing sound effects was accomplished using an MP3 shield from Adafruit. The shield reads .mp3 files from a micro SD card and plays the sound file through one or two speakers. The main problem with the shield is that it requires many of the Arduino's pins to interface properly. This became a major issue with the RedBot, as many of the pins were tied up with the LED matrices, servos, and PIR sensor. Originally SPI was going to be used to interface the shield directly to the RedBot, as SPI pins were labeled on the schematic for the board. However getting the software library included with the shield to work through the SPI pins proved challenging and ultimately futile. The MP3 shield is instead attached to the Uno running its own program for selecting and playing different tracks, and communicates with the RedBot using a two bit address system with an activation line. The activation line tells the Uno when to read the address pins and then play a track based on the address. If further work were to be done on the pumpkin, an I²C serial communication system between the RedBot and the Uno would be preferable over the simple address system used in this design.

PIR Sensor

A PIR sensor is used to set the animatronic pumpkin into motion. The sensor is placed in the bottom of a bowl full of candy embedded in the top of the pumpkin, reading high on the Arduino as long as there is no motion. When someone goes to take candy and disturbs it, the motion sensor will read low, detecting the motion and beginning the main loop. While the PIR sensor accomplishes this task relatively well, the sensor is not perfect. The biggest issue is the delay between when the sensor detects motion and when it is reset. For this project this delay means that the main loop may run multiple times when the sensor is triggered only once. The delay on the sensor is fifteen seconds, so the each pattern in the main loop should be set to last at least fifteen seconds to avoid having multiple patterns played.

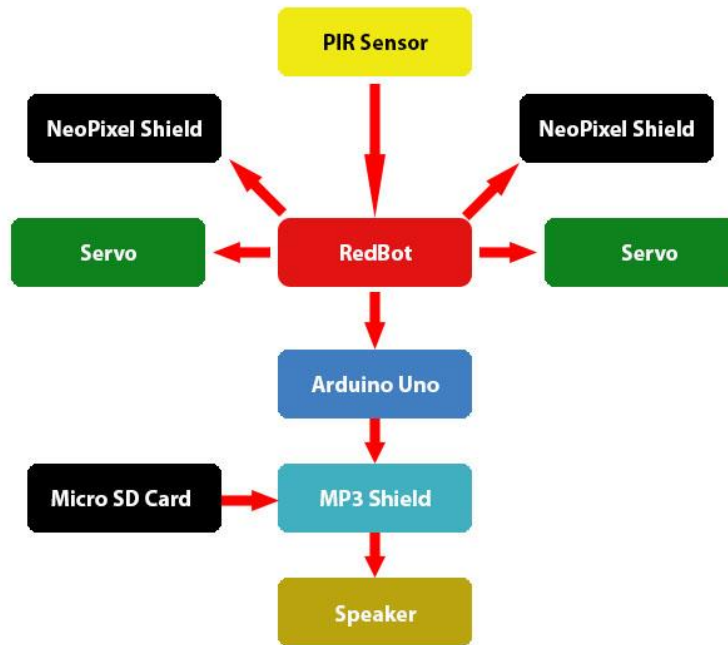


Arms & Jaw (GWS Standard S03N STD Servo Motor)

The arms of the pumpkin are controlled by two servo motors. Servo motors were chosen because of their relatively small size, programming simplicity, and position accuracy. The servo motors allow the arms to move in the desired “robotic motion” for the project. The motors could also be programmed to move to a point, and then return back to the original point with ease, perfect for resetting the arms to a default position after a pattern was completed. Originally during concept generation, stepper motors were considered because of their ability to go to a position and return with great accuracy. Stepper motors proved to be quite large and slow though, which would cause issues installing the motors in the small pumpkin and keeping the motion of the arms fluid. Unfortunately, servo motors generate quite a loud and annoying sound as they move, but this was seen as less of a drawback when compared to the downsides of the stepper. DC motors were also considered, but were quickly ruled out as they are not designed for small, precise movements. The jaw for the pumpkin was originally intended to also be attached to a servo motor to simulate a talking motion, but had to be cut due to time

constraints and issues mounting the servo to the pumpkin body. Further designs could use smaller, quieter servos, since the torque required to move the arms is quite low.

Schematic



The block schematic above shows how the systems of the pumpkin interface with each other. The two external inputs to the system are the PIR motion sensor and the micro SD card that holds the sound files. The PIR sensor detects the motion and informs the RedBot, which then sends the necessary code to the NeoPixel shields, servos, and Arduino Uno. The Uno reads the data supplied by the RedBot and, using the MP3 shield, reads the sound files from the SD card and plays them through the speaker.

Mechanical Design

Mechanical design and construction of the animatronic pumpkin posed a larger issue than originally anticipated. The pumpkin body is composed of a foam craft pumpkin purchased from a craft store. The pumpkin itself proved difficult to work with, being small in size and hard to cut. Several factors had to be changed in order to accommodate the small size of the pumpkin. Parts and wiring needed to be as small as possible to fit inside. This eliminated several parts that were considered in the initial concept generation, like the stepper motors for the arms.

The arms used in the final prototype were 3D printed using the MakerBot Thing-O-Matic in the Learning Factory. The MakerBot is a relatively low resolution 3D printer when compared to the other options available through the university, but were free and convenient to use. The arms did not need much precision when it came to printing either, because they are very basic shapes. The arms were designed using SolidWorks and then exported to the printer using ReplicatorG.

The original idea for the pumpkin included have a jaw that would open and close in sync with the sound to generate a simulation of talking. The jaw would be attached to the top corners of the jaw, where it would have a servo on one corner forcing rotation, and a pin on the other to allow free rotation. Applying this idea to the foam pumpkin proved extremely difficult though. The curvature of the pumpkin, especially towards the bottom, made it difficult to attach the servo and have it correctly oriented. Ideally the servo would be perpendicular to the bottom of the pumpkin, so the jaw would not bind on or hit the bottom of the mouth opening while rotating. In the end the servo could not be installed on the pumpkin properly in enough time, so it had to be cut.

Bill of Materials

Part Name	Price	Quantity
Adafruit "Music Maker" MP3 Shield for Arduino w/3W Stereo Amp - v1.0	\$34.95	1
SanDisk 8GB SD	\$5.95	1
Adafruit NeoPixel Shield for Arduino 40 RGB led matrix	\$27.95	2
PIR Sensor	In Lab	1
Michaels Celebrate It Craft Pumpkin 9"	\$7.49	2
GWS Standard S03N STD Servo Motor	In Lab	3
Sparkfun Redbot Mainboard	In Lab	1
Michaels Black Pail	\$2.95	1
MakerBot 3D Printed Arms	Learning Factory	2
Arduino Uno	In Lab	1
Black Fabric	\$2.00	6" x 6"
8 Ohm Speaker	In Lab	1
Battery Pack	In Lab	1
Total Cost	\$116.73	

Testing

Each electrical component of our project was tested individually as we developed the program. Once we had all the subsystems functioning independently we began combining components one at a time. For example, our LED matrices and servo arms were functioning independently and

then afterwards tested jointly. We found that as we added components we ran into issues providing adequate power or unintentionally using ports that were tied up. After the eyes, arms, and motion sensor were tested together we moved onto the sound. The sound system used a separate Arduino Uno and shield running a second program. We wired the two boards together and tested the address system we used for communication. The issues with the sound system were almost all programming bugs. Through trial and error we were able to effectively communicate between the two boards and the attached mp3 shield.

Issues and Lessons Learned

There are numerous lessons we learned from the process of designing and building our project. The first lesson is that progress isn't linear and sometimes a step backward is necessary. It was very frustrating at times when we spent hours in the lab and didn't accomplish much or ruined the functionality of two subsystems in an effort to combine them. Another piece of advice is to always have a plan B, especially with the least feasible elements of a project. We found ourselves in a difficult situation when our SPI communication between the Arduino Uno and Redbot board wasn't working. Next, it's important to set aside extra time for solving new problems that introduced when integrating subsystems. We found ourselves running out of pins and losing reliable power distribution to components. Lastly, it is impossible to predict when certain electrical components will fail so it is beneficial to have access to a spare. A ruined component can halt progress and in some situations where spares are inaccessible, require a new solution.

Conclusion

Despite setbacks, we are satisfied with the mechatronic jack-o'-lantern we designed and built. It's worth noting that our project encompassed a significant construction aspect. Therefore, we gained some expertise in applying our design knowledge in mechanical, electrical and programming. The motion activated final prototype was able to produce a variety of sound effects, arm movements, and eye patterns. The most valuable experience gained from the project is with integrating subsystems into a final working product. There were a number of technical difficulties that needed to be overcome when combining the individual components into a single system.

If the amount of time for the project was extended, the scope of our project could be scaled accordingly. We would've like to incorporate a functional jaw that was synced to the sound effects produced. Another prospective feature was a mechanism to automatically dispense the candy from the jack-o'-lantern. These two features would add an extra layer of mechanical complexity.

Appendix A: Code

RedBot Board Main Control

```
#include <Servo.h>
#include <Adafruit_GFX.h>
#include <Adafruit_NeoMatrix.h>    //libraries to include
#include <Adafruit_NeoPixel.h>

#define PINL 14    //pins for left and right LED matrices
#define PINR 15
int sensePIN = 18;    //pin for PIR sensor

// Color definitions
#define BLUE 0x001F
#define RED 0xF800
#define GREEN 0x07E0

Adafruit_NeoMatrix matrixL = Adafruit_NeoMatrix(8, 5, PINL,
NEO_MATRIX_TOP + NEO_MATRIX_LEFT +
NEO_MATRIX_ROWS + NEO_MATRIX_PROGRESSIVE,
NEO_GRB + NEO_KHZ800);
Adafruit_NeoMatrix matrixR = Adafruit_NeoMatrix(8, 5, PINR,
NEO_MATRIX_TOP + NEO_MATRIX_LEFT +
NEO_MATRIX_ROWS + NEO_MATRIX_PROGRESSIVE,
NEO_GRB + NEO_KHZ800);

int activate = 19;    //pin that tells sound arduino when to play a track
int add1 = 10;    //first address pin used to select tracks on the sound arduino
int add2 = 11;    //second address pin
int servoL = 16;    //sets left arm servo pin number
int servoR = 17;    //sets right arm servo pin number
int jaw = 9;    //sets jaw servo pin number
Servo left_arm;    //names left servo
Servo right_arm;    //names right servo
int left_pos = 0;    //starting position of 0
int right_pos = 180;    //starting position of 180
int i = 0;    //usable counting variable
int pause = 500;    //delay between commands to servo to avoid interrupting motion
long time = millis(); //set up time variable to determine total time since the program started
long lasttime = millis(); //set up time variable to determine time since last motion for returning to sleep
function

void setup(){
  pinMode(sensePIN,INPUT);
  pinMode(activate,OUTPUT);
```



```
pinMode(add1,OUTPUT);
pinMode(add2,OUTPUT);
Serial.begin(9600);

eye_pattern(5);          //starting sleeping
sound(4);
left_arm.write(left_pos); //resetting arms to default position
right_arm.write(right_pos);
digitalWrite(activate, LOW); //make sure activate pin is low
}

void loop(){
  boolean sensor = digitalRead(sensePIN); //read if there is motion in the candy bucket
  int pattern = random(1,5);             //generate a random number to determine the pattern for animation
  if (sensor == HIGH){                   //if the sensor detects motion a pattern will be played, otherwise it will
do no movement and sit at the last eye pattern
    if (pattern == 1){
      sound(1);                          //each pattern is comprised of a sound clip, LED eye pattern, and arm servo
movement set
      eye_pattern(1);
      arm_move(1);
      digitalWrite(activate,LOW);
    }
    if (pattern == 2){
      sound(2);
      eye_pattern(2);
      arm_move(2);
      digitalWrite(activate,LOW);
    }
    if (pattern == 3){
      sound(3);
      eye_pattern(3);
      arm_move(3);
      digitalWrite(activate,LOW);
    }
    if (pattern == 4){
      sound(3);
      eye_pattern(4);
      arm_move(1);
      digitalWrite(activate,LOW);
    }
    lasttime = millis();
  }
  delay(2000);
  time = millis();
  if (time - lasttime > 20000){ //returns pumpkin to sleep mode if the last action was more than twenty
seconds ago
    eye_pattern(5);
```

```
    sound(4);
    digitalWrite(activate,LOW);
  }
}

void eye_pattern(int pattern){ //sets eye pattern currently on display
  int x = matrixL.width();
  int pass = 0;

  matrixL.begin();
  matrixL.setTextWrap(false);
  matrixL.setBrightness(100);
  matrixL.fillScreen(0);          //setup for the left matrix
  matrixL.setCursor(x,0);

  matrixR.begin();
  matrixR.setTextWrap(false);      //setup for right matrix
  matrixR.setBrightness(100);
  matrixR.fillScreen(0);
  matrixR.setCursor(x,0);

  if (pattern == 1){              //neutral
    matrixL.drawFastHLine(1,4,7,BLUE);
    matrixL.drawPixel(1,3,BLUE);
    matrixL.drawLine(1,2,3,0,BLUE);
    matrixL.drawLine(4,0,7,3,BLUE);
    matrixL.drawPixel(4,2,RED);

    matrixR.drawFastHLine(0,4,7,BLUE);
    matrixR.drawPixel(6,3,BLUE);
    matrixR.drawLine(6,2,4,0,BLUE);
    matrixR.drawLine(3,0,0,3,BLUE);
    matrixR.drawPixel(3,2,RED);
  }
  if (pattern == 2){              //angry boxy
    matrixL.drawFastVLine(0,0,5,GREEN);
    matrixL.drawFastHLine(0,4,8,GREEN);
    matrixL.drawFastHLine(0,0,4,GREEN);
    matrixL.drawLine(4,1,6,3,GREEN);
    matrixL.drawPixel(3,2,RED);

    matrixR.drawFastVLine(7,0,5,GREEN);
    matrixR.drawFastHLine(0,4,8,GREEN);
    matrixR.drawFastHLine(4,0,4,GREEN);
    matrixR.drawLine(3,1,1,3,GREEN);
    matrixR.drawPixel(4,2,RED);
  }
}
```

```
if (pattern == 3){          //angry rounded
  matrixL.drawFastHLine(2,4,5,GREEN);
  matrixL.drawPixel(2,3,GREEN);
  matrixL.drawFastVLine(1,1,3,GREEN);
  matrixL.drawFastHLine(2,0,3,GREEN);
  matrixL.drawLine(5,1,7,3,GREEN);
  matrixL.drawRect(3,2,2,2,RED);

  matrixR.drawFastHLine(1,4,5,GREEN);
  matrixR.drawPixel(5,3,GREEN);
  matrixR.drawFastVLine(6,1,3,GREEN);
  matrixR.drawFastHLine(3,0,3,GREEN);
  matrixR.drawLine(2,1,0,3,GREEN);
  matrixR.drawRect(3,2,2,2,RED);
  //matrixR.drawPixel(3,2,RED);
}
if (pattern == 4){          //skeptical
  matrixL.drawFastHLine(2,1,4,BLUE);
  matrixL.drawFastHLine(2,3,4,BLUE);
  matrixL.drawPixel(1,2,BLUE);
  matrixL.drawPixel(6,2,BLUE);
  matrixL.drawFastHLine(3,2,2,RED);

  matrixR.drawFastHLine(2,1,4,BLUE);
  matrixR.drawFastHLine(2,3,4,BLUE);
  matrixR.drawPixel(1,2,BLUE);
  matrixR.drawPixel(6,2,BLUE);
  matrixR.drawFastHLine(3,2,2,RED);
}
if (pattern == 5){          //sleeping
  matrixL.drawFastHLine(1,2,6,BLUE);
  matrixL.drawFastHLine(2,3,4,BLUE);

  matrixR.drawFastHLine(1,2,6,BLUE);
  matrixR.drawFastHLine(2,3,4,BLUE);
}
matrixL.show(); //displays the current set on the LED matrices
matrixR.show();
}

void arm_move (int set){    //moves servo arms depending on movement set input
  left_arm.attach(servoL);
  right_arm.attach(servoR);
  if (set == 1){           //walking
    while (i<3){
      left_arm.write(0);
      right_arm.write(90);
      delay(pause);
    }
  }
}
```

```
    left_arm.write(90);
    right_arm.write(180);
    delay(pause);
    i++;
  }
}
if (set == 2){          //arms up
  left_arm.write(90);
  right_arm.write(90);
  delay(2000);
}
if (set == 3){          //pointing
  left_arm.write(120);
  delay(2000);
}
left_arm.write(left_pos); //resetting to default position
right_arm.write(right_pos);
i = 0;
delay(pause);
}

void sound (int track){          //function to communicate with the second arduino running the music
shield
  if (track == 1){
    digitalWrite(add1, LOW);      //address system for the four tracks
    digitalWrite(add2, LOW);
    digitalWrite(activate,HIGH);  //activate line to tell the sound arduino to read the address pins
  }
  if (track == 2){
    digitalWrite(add1, HIGH);
    digitalWrite(add2, LOW);
    digitalWrite(activate,HIGH);
  }
  if (track == 3){
    digitalWrite(add1, LOW);
    digitalWrite(add2, HIGH);
    digitalWrite(activate,HIGH);
  }
  if (track == 4){
    digitalWrite(add1, HIGH);
    digitalWrite(add2, HIGH);
    digitalWrite(activate,HIGH);
  }
}
```

Arduino Uno Sound Control

```
// include SPI, MP3 and SD libraries
#include <SPI.h>
#include <Adafruit_VS1053.h>
#include <SD.h>

// These are the pins used for the breakout example
#define BREAKOUT_RESET 9 // VS1053 reset pin (output)
#define BREAKOUT_CS 10 // VS1053 chip select pin (output)
#define BREAKOUT_DCS 8 // VS1053 Data/command select pin (output)
#define SHIELD_CS 7 // VS1053 chip select pin (output)
#define SHIELD_DCS 6 // VS1053 Data/command select pin (output)
#define CARDCS 4 // Card chip select pin
#define DREQ 3 // VS1053 Data request, ideally an Interrupt pin

Adafruit_VS1053_FilePlayer musicPlayer =
  Adafruit_VS1053_FilePlayer(SHIELD_CS, SHIELD_DCS, DREQ, CARDCS);

int add1 = 14;
int add2 = 15;
int add3 = 16;
int actPin = 17;

void setup() {
  pinMode(actPin, INPUT);
  pinMode(add1, INPUT);
  pinMode(add2, INPUT);
  pinMode(add3, INPUT);

  Serial.begin(9600);
  musicPlayer.begin();

  SD.begin(CARDCS);
  // Set volume for left, right channels. lower numbers == louder volume!
  musicPlayer.setVolume(5,5);
}

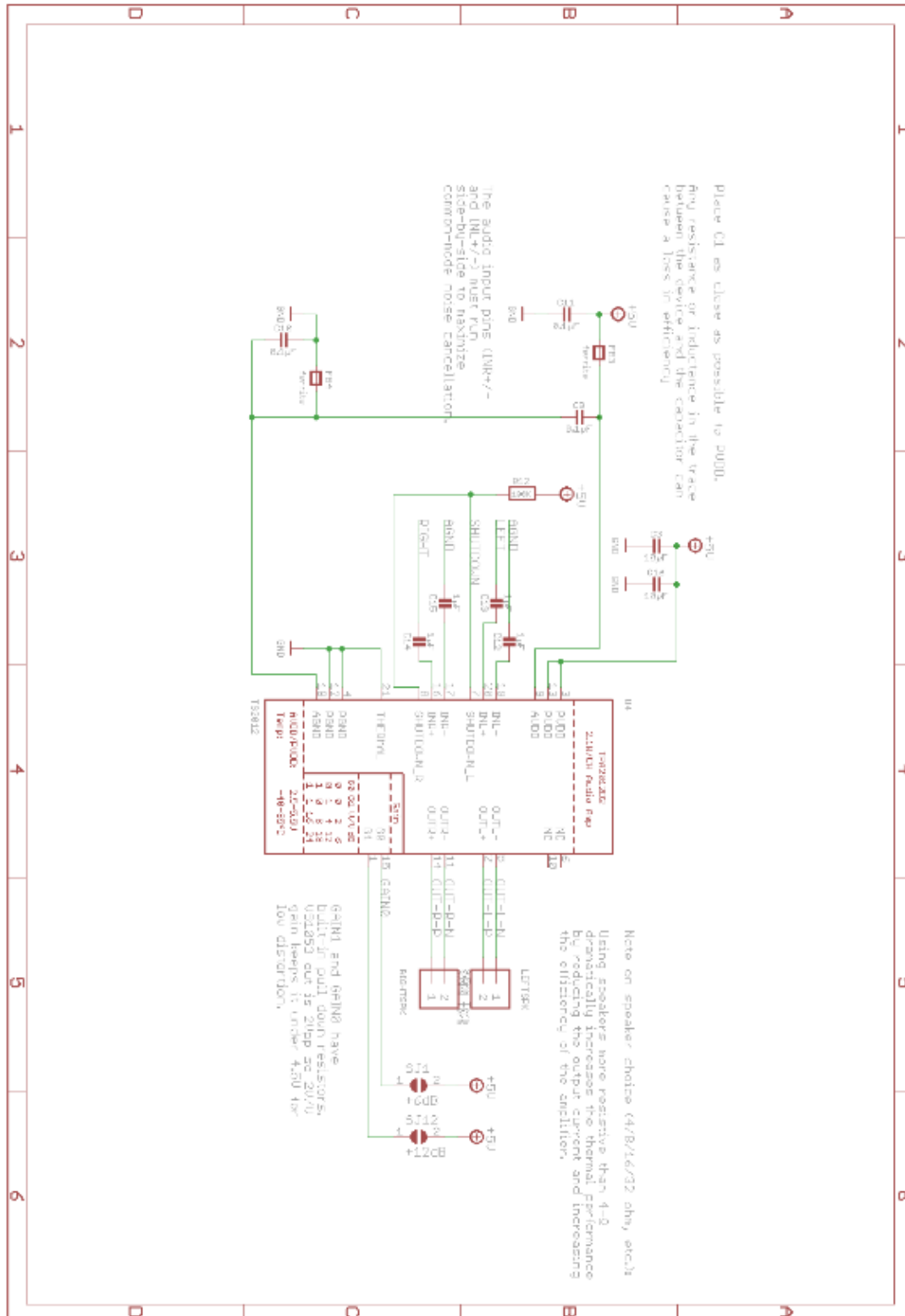
void loop(){
  int activate = digitalRead(actPin);
  Serial.print(activate);
  if (activate == HIGH){
    int num1 = digitalRead(add1);
```

```
int num2 = digitalRead(add2);
int num3 = digitalRead(add3);
int pattern = num1 + (2*num2) + (4*num3);

if (pattern == 0){
    musicPlayer.playFullFile("track1.mp3");
    musicPlayer.stopPlaying();
}
if (pattern == 1){
    musicPlayer.playFullFile("track2.mp3");
    musicPlayer.stopPlaying();
}
if (pattern == 2){
    musicPlayer.playFullFile("track3.mp3");
    musicPlayer.stopPlaying();
}
if (pattern == 3){
    musicPlayer.playFullFile("track4.mp3");
    musicPlayer.stopPlaying();
}
else{
    musicPlayer.stopPlaying();
}
Serial.println(pattern);
}
}
```

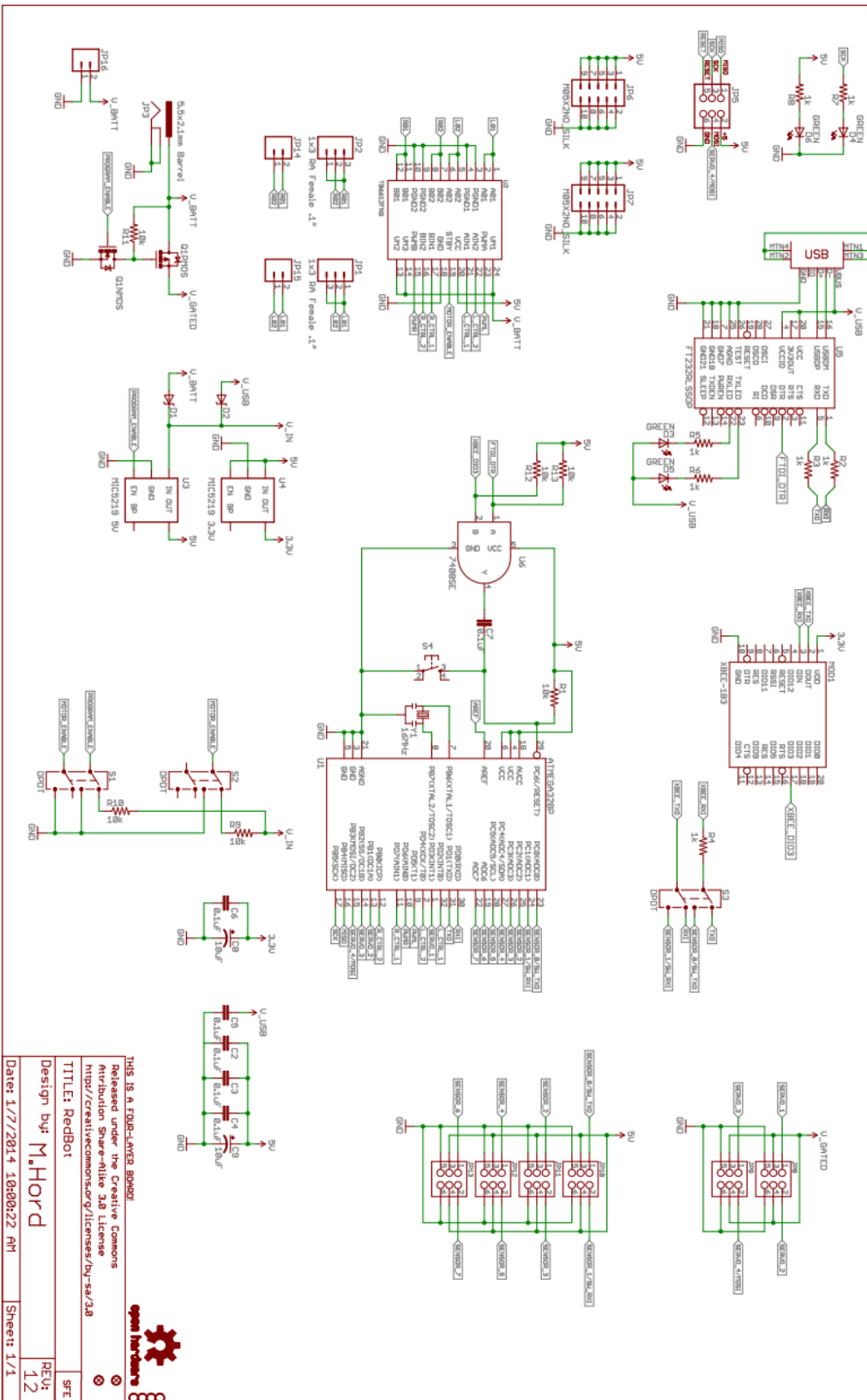
Appendix B: Data Sheets and Schematics

Adafruit MP3 Shield





RedBot Mainboard



GWS Standard S03N STD Servo Motor

GWS S03N - Standard Servo

Basic Information

Modulation:	Analog
Torque:	4.8V: 47.0 oz-in (3.38 kg-cm) 6.0V: 56.0 oz-in (4.03 kg-cm)
Speed:	4.8V: 0.23 sec/60° 6.0V: 0.18 sec/60°
Weight:	1.45 oz (41.1 g)
Dimensions:	Length: 1.56 in (39.6 mm) Width: 0.79 in (20.1 mm) Height: 1.40 in (35.6 mm)



PIR Motion Sensor



Web Site: www.parallax.com
 Forums: forums.parallax.com
 Sales: sales@parallax.com
 Technical: support@parallax.com

Office: (916) 624-8333
 Fax: (916) 624-8003
 Sales: (888) 512-1024
 Tech Support: (888) 997-8267

PIR Sensor (#555-28027)

General Description

The PIR (Passive Infra-Red) Sensor is a pyroelectric device that detects motion by measuring changes in the infrared levels emitted by surrounding objects. This motion can be detected by checking for a high signal on a single I/O pin.

Features

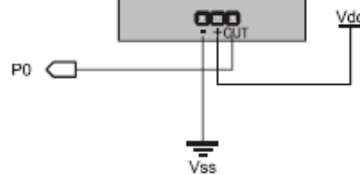
- Single bit output
- Small size makes it easy to conceal
- Compatible with all Parallax microcontrollers
- 3.3V & 5V operation with <100uA current draw

Application Ideas

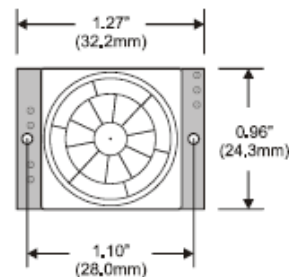
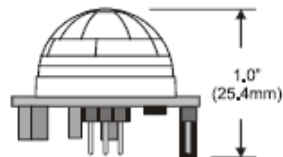
- Alarm Systems
- Halloween Props

Quick Start Circuit

Note: The sensor is active high when the jumper (shown in the upper left) is in either position.



Module Dimensions



Theory of Operation

Pyroelectric devices, such as the PIR sensor, have elements made of a crystalline material that generates an electric charge when exposed to infrared radiation. The changes in the amount of infrared striking the element change the voltages generated, which are measured by an on-board amplifier. The device contains a special filter called a Fresnel lens, which focuses the infrared signals onto the element. As the ambient infrared signals change rapidly, the on-board amplifier trips the output to indicate motion.

Pin Definitions and Ratings

Pin	Name	Function
-	GND	Connects to Ground or Vss
+	V+	Connects to Vdd (3.3V to 5V) @ ~100uA
OUT	Output	Connects to an I/O pin set to INPUT mode (or transistor/MOSFET)

Jumper Setting

Position	Mode	Description
H	Retrigger	Output remains HIGH when sensor is retriggered repeatedly. Output is LOW when idle (not triggered).
L	Normal	Output goes HIGH then LOW when triggered. Continuous motion results in repeated HIGH/LOW pulses. Output is LOW when idle.

Connecting and Testing

Connect the 3-pin header to your circuit so that the minus (-) pin connects to ground or Vss, the plus (+) pin connects to Vdd and the OUT pin connects to your microcontroller's I/O pin. One easy way to do this would be to use a standard servo/LCD extension cable, available separately from Parallax (#805-00002). This cable makes it easy to plug sensor into the servo headers on our Board Of Education or Professional Development Board. If you use the Board Of Education, be sure the servo voltage jumper (located between the 2 servo header blocks) is in the Vdd position, not Vin. If you do not have this jumper on your board you should manually connect to Vdd through the breadboard. You may also plug the sensor directly into the edge of the breadboard and connect the signals from there. Remember the position of the pins when you plug the sensor into the breadboard.

Calibration

The PIR Sensor requires a 'warm-up' time in order to function properly. This is due to the settling time involved in 'learning' its environment. This could be anywhere from 10-60 seconds. During this time there should be as little motion as possible in the sensors field of view.

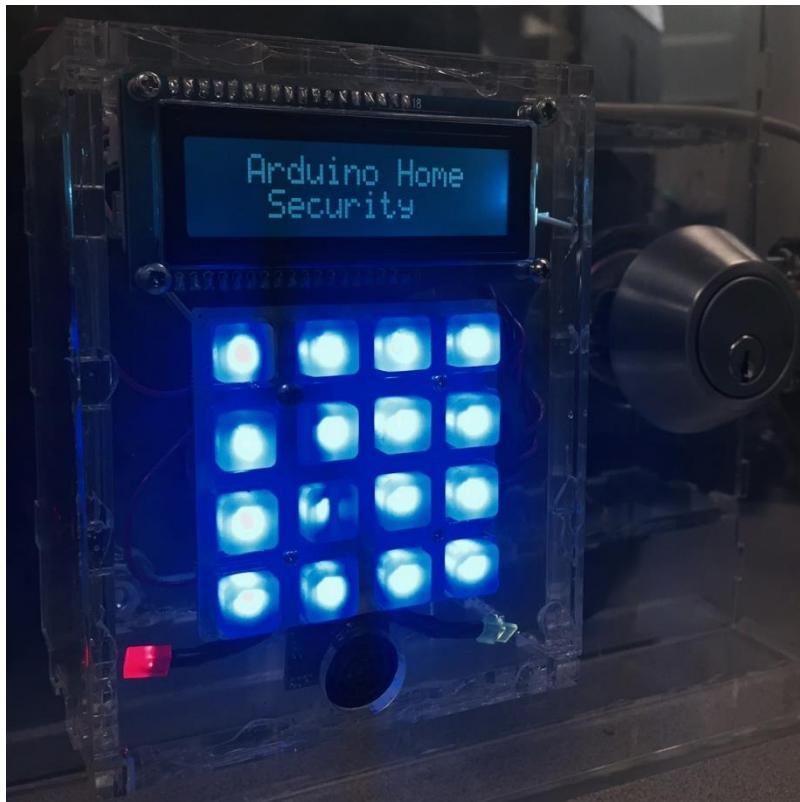
Sensitivity

The PIR Sensor has a range of approximately 20 feet. This can vary with environmental conditions. The sensor is designed to adjust to slowly changing conditions that would happen normally as the day progresses and the environmental conditions change, but responds by making its output high when sudden changes occur, such as when there is motion.

Resources and Downloads

Check out the PIR Sensor product page for example programs and more:

http://www.parallax.com/detail.asp?product_id=555-28027



Arduino Home Security

PASSWORD BASED SECURITY SYSTEM

ME 445 FALL 2014

DAN GIVIGLIANO & SHRUTI MOTIWALE



Contents

Introduction:	2
Components:	3
MaxSonar:	3
Trellis Keypad:	3
Liquid Crystal Display:	3
Hitec HS-805BGB Servo:	3
Magnetic Contact Switch:	4
Arduino Wifi Shield:	4
Camera:	4
Speakers:	4
Exterior Design	5
System Design:	5
Motion Detection:	5
Activation:	5
Password Entry:	6
Unlocking the System:	6
Additional Features:	6
Incorrect Password:	6
Away Messages:	6
Website Features:	7
Pushingbox.com:	7
Problems Encountered	7
Conclusion	8
Acknowledgement:	9
Appendices	9
Appendix A: Bill of Materials	9
Appendix B: References	9
Appendix C: Specifications	10
Appendix D: Arduino Code	13
MAIN CODE:	13
FUNCTIONS:	20
Functions for Password:	20

Introduction:

This semester in ME445 fall 2014 our goal was to challenge ourselves and apply what we have learned throughout the semester about the interfacing of electro-mechanical systems to microcomputers for data acquisition, data analysis and digital control.

Applying the concepts learned throughout the course, we designed, built, and programed an automated door unlocking security system using an Arduino Uno microcontroller. The system was designed to give the user a wide array useful features from a simple password entry code to high security remote monitoring. This project was designed to be extremely versatile in order to accommodate individuals for personal use as well as companies and other organizations.

The main objective of our automated unlocking system was to incorporate a keypad that required a password for the door to unlock. Once the password was entered the door would be unlocked from the other side using a servo to turn a deadbolt. From there we were able to add many features we felt would be beneficial for the user.

The features we added started with the sonar detection sensor. This was applied to keep the system running constantly and activate/deactivate the system only when need. From there we added a liquid crystal display that would display messages/prompts for the user to guide them through the process effortlessly.

Once the basis of our system was complete the next step was to integrate a more advance security system by adding system self-monitoring and wireless control. This required internet connectivity and a server to manage the data recorded by the Arduino. Using the internet the system was able to monitor the devices attached to Arduino and send notifications to the owner of the account for many different scenarios. The system would also be able to store records of the systems activity in a cloud. With wireless control the user would have the ability to remotely control the system or change settings remotely from their phone or computer.

Lastly our goal was to incorporate a camera that would work with the motion sensor to detect and record activity outside of the door. The camera would be activated by the motion and could record a video or snap a picture of the person outside of the door. The activity would then be time stamped and saved in the system or sent to the cloud. This would be very useful for those seeking a more advanced security system.

Components:

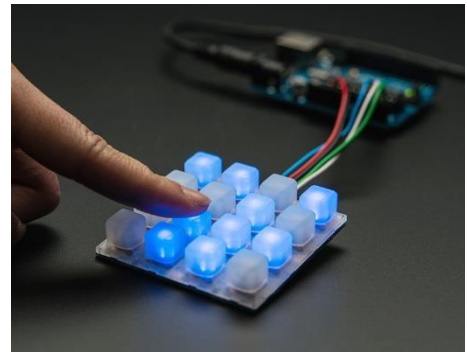
MaxSonar:

The Maxbotix Maxsonar is an ultrasonic range finder that can determine the distance to an object in front of it using ultrasonic wave pulses. This sonar has a maximum range of about 21 feet and has a precision of approximately 1 in. The sonar reads the distance at a rate of 20Hz. The sensor works by sending an ultrasonic waves outward and then listens for an echo after the wave has reflected off an object. As the wave is sent and then again when the echo recorded the sensor sends pulses to the Arduino. The time between each pulse can be converted into distance base on the speed of sound.



Trellis Keypad:

The Trellis PCB by AdaFruit is a backlight keypad driver system built to match a 4x4 elastomer key pad. The connections are multiplexed so each button is uniquely addressed and has its own LED below, built into the board. Below the rubber buttons are built in sensors that can detect whenever the button is pressed. The Trellis keypad can be easily programed with an Arduino. Unlike most keypad the Trellis only uses 3-4 pins making it more convenient for our project.



Liquid Crystal Display:

AdaFruit's 16x2 RGB backlight LCD and TTL serial backpack kit was perfect for displaying the messages we wanted on LCD screen. The backpack fit on the back of the standard 16x2 LCD display and was very helpful reducing the number of pins needed and simplifying our code.



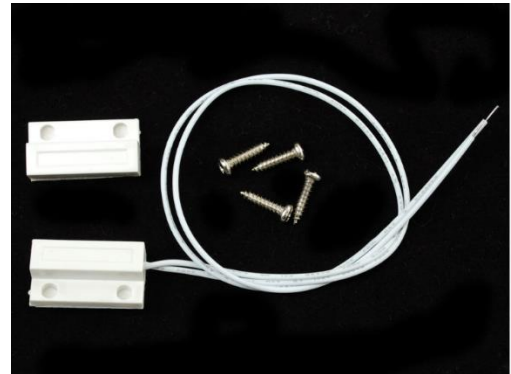
Hitec HS-805BGB Servo:

The 3-pole Hitec Servo used in our project was compatible of rotating 180 degrees with a pulse cycle of 20ms. This servo was used because it could provide a torque of 275 oz-in at 4.8V, which was more than enough power to open a deadbolt with ease. When connected to the Arduino we were able to control this rotation with high precision by using pulse-width modulation.



Magnetic Contact Switch:

This sensor has two parts to it, a reed switch and a magnet, both encased in ABS plastic. The reed switch would be attached to the edge of the door. The magnet side would need to lineup across from the reed switch and be attached within 13mm of it when the door is closed. When the door is closed the magnet will close the reed switch by connecting the two wires inside. Once the door is open the wires will disconnect and the reed switch will be open. Using the Arduino to read the voltage we can detect when the door is open and closed.



Arduino Wifi Shield:

We decided to use the WiFi shield for incorporating wireless connectivity in our project. We came across the term IoT, or Internet of Things, which is a concept where each device or sensor is assigned a unique identifier and this is used to transfer data over internet without any direct human intervention. We tried a number of websites which provide platforms to developers for doing this, like Xively.com, pushingbox.com, temboo.com, m2x.att.com, data.sparkfun.com, grovestreams.com etc. and decided to go for two of these, namely pushingbox.com and temboo.com, primarily for the variety of ways they offer to log the data, get notifications and control the device as well.



Camera:

Although we did not get the camera to work there are many that can be used for this project. The camera we intended to use was the Adafruit Serial JPEG Camera with NTSC video. This camera is used in many security systems and outputs NTSC monochrome video and can take snapshots of that video (in color) and transmit them over the TTL serial link. These images are compressed JPEG and can be stored easily on a SD card. This camera can also replace because it has a built in motion detection.



Speakers:

Using a piezo speaker you can easily make buzzes, play music notes, or even simple melodies. For our project we used simple RTTL (Ring Tone Text Transfer Language) to create animated melodies. Ours consist mainly of just simple musical note and Mario theme music. In order to play more advance sounds an mp3 shield would be needed.



Exterior Design

Many factors went into the project's overall design. The project first had to be able to function properly so mounting the servo onto the deadbolt securely was extremely important. The design also needed to be very versatile so it can be easily attached to a multitude of doors. The last factor we considered was its appearance. Since the system was going to be mounted on people's doors it needed to be sleek, compact, and appealing for the user.

The system consisted of two components, one for the servo inside and the other for the LCD display and Trellis key pad on the outside of the door. In order to create a durable casing for both we used clear quarter inch thick acrylic. This acrylic was laser cut so that the pieces could interlock and fit together securely. The casing for the inside was built to hold the servo firmly in place can be lined up with deadbolt and then easily attached to the door securely. Attached to the servo we laser cut an acrylic head that would fit securely around a typical deadbolt. Different heads can be attached freely for different shaped deadbolts.

The casing on the inside of the door was laser cut so the LCD display, Trellis keypad, and sonar would fit firmly in place. The casing was designed with the tightly interlocking acrylic pieces. When attached to the door the casing can only be taken off from the inside of the door.

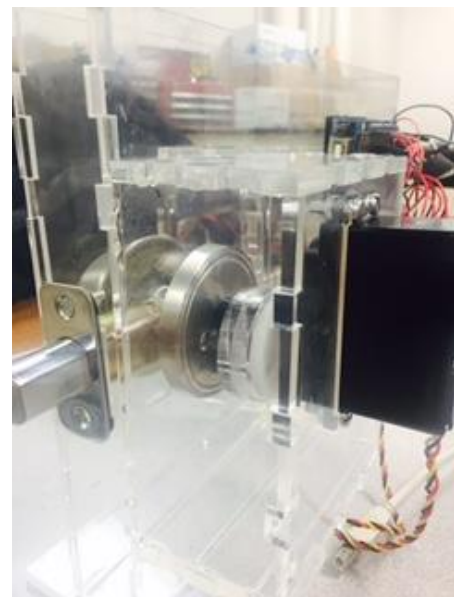
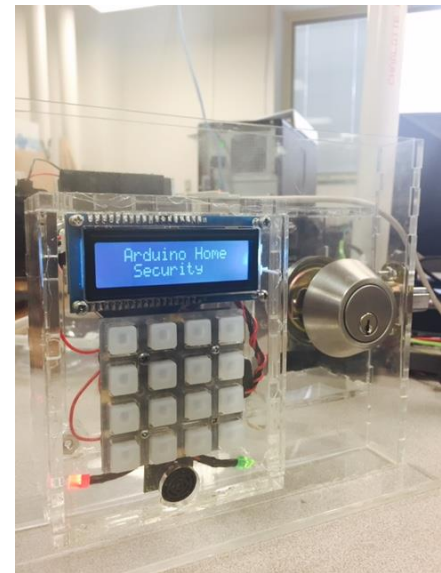
System Design:

Motion Detection:

Using the Maxbotix MaxSonar ultrasonic rangefinder we were able to record reliable range readings outside of the door. This rangefinder provided a stable and precise reading with a low power demand. The sonar was programmed to continuously record the range outside of the door. As a person approached the door the motion detection system would detect them and activate the rest of the system. In order to filter the readings recorded by the sonar the Arduino would only activate the system when the sonar detected motion coming toward the door and the person was within a specific specific range in of the door. Visible feedback was provided by an LED indicator that would blink once motion was detected. Once the motion met the criteria established in the code the Arduino would activate the rest of the system.

Activation:

Upon activation, the button on Trellis keypad would light up in an entertaining pattern and play a sequence of tones. The LCD display would also light up and display a greeting for the user. This process would signify



that the system has been activated. LCD screen would then prompt the user to enter their username or password.

Password Entry:

Using the Trellis keypad the user will then enter their password. For a system with multiple users the Arduino can be programmed with different passwords for each user. If the user is a guest they would just press a specified button that would then notify the owner via text or phone call that their guest has arrived. While entering the password on the keypad LED will light up upon being pressed while playing a short tune from the speakers to provide sensory feedback. Each button pressed is recorded by the Arduino and compared to the password(s) saved in the system. If the sequence of buttons pressed matches a save password the system will welcome the specific user on the LCD screen.

Unlocking the System:

Once the password has been confirmed a melody will be played (Ours was the Super Mario Brother's theme) and the servo attached to the lock will be activated. The LCD display will display the message "Unlocking. . ." while the system is in the process of unlocking. A red LED indicator will also be lit while the system is locked. While unlocking the Trellis keypad LEDs will light up in a complete circular motion which corresponds to the servos rotation. Once the servo has rotated the deadbolt completely the door is now unlocked and the LCD display will display "Unlocked" and a green LED will be lit replacing the red to signify the system is ready to be accessed.

While not incorporated in our current design. A magnetic contact switch should be attached to the edge of the door and the frame across from it. Once the door is opened the magnetic switch would trigger the Arduino to record the current time. If the door is not shut within a specific time a buzzer will go off notifying the user that the door is ajar. If the door continues to remain open the owner of the account can be notified by text or call. Once the door is shut and the magnetic contact switch is connected the servo will again be activated and turn the deadbolt into the locked position.

Additional Features:

Incorrect Password:

In order to keep people from trying to guess the password, the system will only allow and set number of failed attempts before shutting down. The owner can change the settings as they wish. The system can shut down for a specific period of time or it can shut down completely until the owner reactivates it. When the system shuts down the owner will immediately get a text to the phone with a picture of the user trying to enter. They can also get a call from to their phone that will give them the option to reactivate the system or keep it locked by pressing one or two.

Away Messages:

Depending on who uses this device this feature can be used in many different ways. Our away message feature can be applied by the user to display a message on the LCD display when someone arrive at the door.

Website Features:

Temboo.com:

Using this website we incorporated features like getting an email notification, text messages, as well as phone calls.

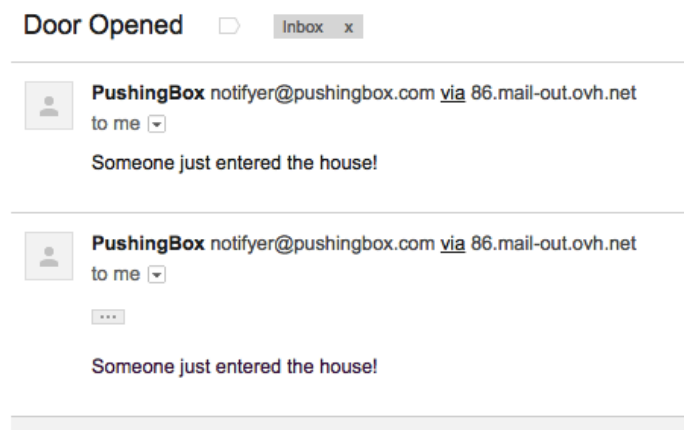
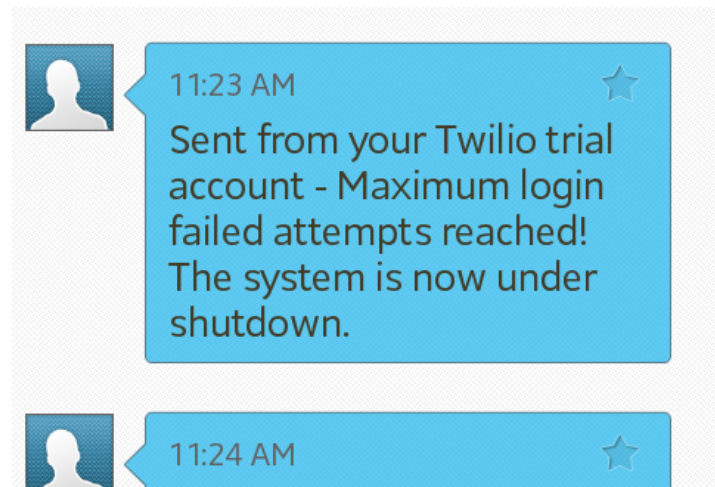
Each such notification type is described by a unique identifier, and gets activated based on the value on a particular analog or digital pin. We sent a HIGH to a particular digital pin whenever an instance occurred, eg. HIGH on D10 if someone entered, HIGH on D11 if the system shuts down etc. Whenever the code detects this, it runs the routine to make a phone call or send a text message to the owner.

This website also allows us to connect back to the system. Whenever someone is waiting at the door, they can press a button on the keypad, which will make a phone call to the owner. The phone call informs the owner that someone is waiting at the door, and gives the owner an option to press a button to lock or unlock the door. The camera can take a picture of the person and send it to the owner, based on which the owner can decide whether to unlock the door or not. Although we couldn't get the camera to work, if a camera is added, it is very easy to add this functionality. Moreover, an image with a timestamp can be easily uploaded to a spreadsheet in the form of a log containing who tried to enter the house at what time.

Although this website gives us a lot more features, it is less helpful from learning point of view for a course project, as it is more or less GUI based, and has little scope for playing around in the code.

Pushingbox.com:

We also tried another website, pushingbox.com, which gives us a lot more freedom to tweak the code according to our requirements. The calling and text message feature is not available with this website, but similar functionalities can be achieved with the apps which can be installed in any cell phone. One can also get notifications on their web browsers. There are two parts to the code - scenario and service. A scenario is the action which triggers the call, like door being unlocked, and a service is the notification which must be sent, like a mail or a notification on an iPhone or a web browser



Problems Encountered

This project incorporated a lot of different components which lead to a lot of problems integrating them all together. After added more and more features the code was getting difficult to debug when things were

not working. This was later fixed by simplifying the code by creating many functions. We also organized it in many tabs, and adding comments were ever we could. We also saved multiple codes once we had it function correctly.

Another big problem we ran into was importing libraries. In order to play tones, and use the servo we could not use the servo library because they had conflicting timers. We worked around this by making our own functions to control the servo.

We also encountered problems several times with loose or poor connections due to bad wires or poor soldering (mostly poor soldering). Using the millimeter we found the source much faster and were able to correct the problem.

Wireless control ate more of our time than the entire rest of the project. Initially we were using a WiFi shield from adafruit.com for connecting to the internet. The code for this WiFi shield was too large for an Arduino Uno, and we had to move to an Arduino Mega which was already available in the lab. But after spending a lot of time on getting it to work, we realized that there was apparently something wrong with the soldering as the power lines were heating up too much. Buying the shield from Adafruit itself was a wrong choice; but we fell for the exceedingly low price. We later on found out that the board is launched very recently, and their libraries are not robust yet. They admit in their forums and comment sections of the code that most libraries have bugs in them and they don't work as expected. We should have done a more thorough research before ordering the board.

Then we decided to move to an Ethernet shield which was already available in the lab. We tried this with the Internet at my home(since the institute Ethernet is blocked), and it worked perfectly. But on fixing everything else on the mega, we realized that the components were behaving strangely (flickering LEDs on the trellis keypad, strange noises from the speaker and jittery motor), while they behaved perfectly fine on the Uno. We concluded that there must be something wrong with the power supply on the mega, since the internet seemed to be working fine on it. The Ethernet board also derives some power from the cable, so it worked fine, but everything else couldn't. So we decided to keep all the components on the Uno, and run the internet through the Mega board, and communicate through a chip. If we had spent less time on the WiFi shield, we probably would have had more time and could have ordered another board, which could have helped us to avoid such unnecessary complications.

Conclusion

Overall the project was a success. Although we did not get every component we wanted working for a high level of security we still had a very reliable functioning system that is far superior to that at the lab. With more time we could have easily gotten every feature we wanted and more.

The reason both of us took this class was because we knew very little about electronics and microcontrollers. Everything we learned from class and from doing research on this project helped us discover so many things we did not know before about electronics and computer programming methods.

The project taught us more than just how to apply electro-mechanical systems to micro-controllers. Throughout the semester we developed many great trouble shooting methods. With an endless amount of mistakes in our code and wiring we became experts in debugging computer code and circuits. Coming out of this semester we will see everything completely different. It opened the door to a whole new world of electronics.

Acknowledgement:

We would like to thank Dr. Sommer for this class and providing us the opportunity, knowledge and resources to do this project. We are also extremely grateful to Mike for the support throughout the project and for answering our endless emails and queries.

Appendices

Appendix A: Bill of Materials

Component	Vender	Cost
Arduino Uno	Amazon	14.95
Wifi Shield	AdaFruit	39.95
LCD Display+Backpack	AdaFruit	24.95
Trellis Key Pad	AdaFruit	9.95
MaxSonar	AdaFruit	24.95
Servo Motor	AdaFruit	14.00
Magnetic Contact Switch	AdaFruit	3.95
Speakers	AdaFruit	1.50
Camera	AdaFruit	39.95
Other (Acrylic,Screw,ect)	McMaster-Car	10.00

Appendix B: References

<https://learn.adafruit.com/adafruit-trellis-diy-open-source-led-keypad>

<https://temboo.com/>

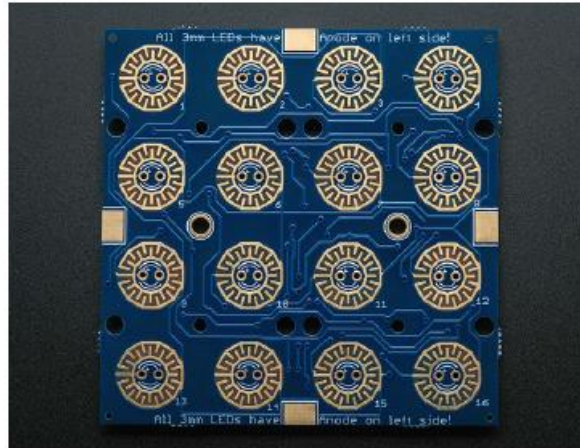
<https://www.pushingbox.com/>

<http://forum.arduino.cc/>

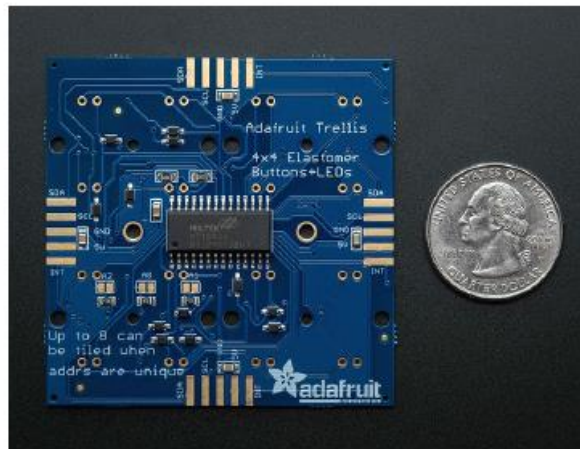
<http://www.linuxcircle.com/2013/03/31/playing-mario-bros-tune-with-arduino-and-piezo-buzzer>

Appendix C: Specifications

TRELLIS:



The Trellis PCB is specially made to match the Adafruit 4x4 elastomer keypad. Each Trellis PCB has 4x4 pads and 4x4 matching spots for 3mm LEDs. The circuitry on-board handles the background key-presses and LED lighting for the 4x4 tile. However, it does not have any microcontroller or other 'brains' - an Arduino (or similar microcontroller) is required to control the Trellis to read the keypress data and let it know when to light up LEDs as desired.



Each tile has an I2C-controlled LED sequencer and keypad reader already on it. The chip can control all 16 LEDs individually, turning them on or off. It cannot do grayscale or dimming. The same chip also reads any keypresses made with the rubber keypad. The connections are 'diode multiplexed' so you do not have to worry about "ghosting" when pressing multiple keys, each key is uniquely addressed.

LV-MaxSonar® -EZ™ Series

High Performance Sonar Range Finder

MB1000, MB1010, MB1020, MB1030, MB1040

With 2.5V - 5.5V power the LV-MaxSonar-EZ provides very short to long-range detection and ranging in a very small package. The LV-MaxSonar-EZ detects objects from 0-inches to 254-inches (6.45-meters) and provides sonar range information from 6-inches out to 254-inches with 1-inch resolution. Objects from 0-inches to 6-inches typically range as 6-inches¹. The interface output formats included are pulse width output, analog voltage output, and RS232 serial output. Factory calibration and testing is completed with a flat object. ¹See Close Range Operation



Features

- Continuously variable gain for control and side lobe suppression
- Object detection to zero range objects
- 2.5V to 5.5V supply with 2mA typical current draw
- Readings can occur up to every 50mS, (20-Hz rate)
- Free run operation can continually measure and output range information
- Triggered operation provides the range reading as desired
- Interfaces are active simultaneously
- Serial, 0 to Vcc, 9600 Baud, 81N
- Analog, (Vcc/512) / inch
- Pulse width, (147uS/inch)

- Learns ringdown pattern when commanded to start ranging
- Designed for protected indoor environments
- Sensor operates at 42KHz
- High output square wave sensor drive (double Vcc)

Benefits

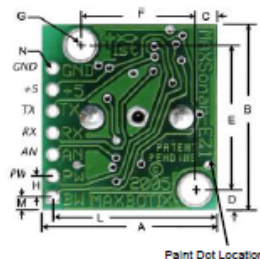
- Very low cost ultrasonic rangefinder
- Reliable and stable range data
- Quality beam characteristics
- Mounting holes provided on the circuit board
- Very low power ranger, excellent for multiple sensor or battery-based systems
- Fast measurement cycles

- Sensor reports the range reading directly and frees up user processor
- Choose one of three sensor outputs
- Triggered externally or internally

Applications and Uses

- UAV blimps, micro planes and some helicopters
- Bin level measurement
- Proximity zone detection
- People detection
- Robot ranging sensor
- Autonomous navigation
- Multi-sensor arrays
- Distance measuring
- Long range object detection
- Wide beam sensitivity

LV-MaxSonar-EZ Mechanical Dimensions



A	0.785"	19.9 mm	H	0.100"	2.54 mm
B	0.870"	22.1 mm	J	0.610"	15.5 mm
C	0.100"	2.54 mm	K	0.645"	16.4 mm
D	0.100"	2.54 mm	L	0.735"	18.7 mm
E	0.670"	17.0 mm	M	0.065"	1.7 mm
F	0.510"	12.6 mm	N	0.038" dia.	1.0 mm dia.
G	0.124" dia.	3.1 mm dia.	weight, 4.3 grams		

Part Number	MB1000	MB1010	MB1020	MB1030	MB1040
Paint Dot Color	Black	Brown	Red	Orange	Yellow



Close Range Operation

Applications requiring 100% reading-to-reading reliability should not use MaxSonar sensors at a distance closer than 6 inches. Although most users find MaxSonar sensors to work reliably from 0 to 6 inches for detecting objects in many applications, MaxBotix® Inc. does not guarantee operational reliability for objects closer than the minimum reported distance. Because of ultrasonic physics, these sensors are unable to achieve 100% reliability at close distances.

Warning: Personal Safety Applications

We do not recommend or endorse this product be used as a component in any personal safety applications. This product is not designed, intended or authorized for such use. These sensors and controls do not include the self-checking redundant circuitry needed for such use. Such unauthorized use may create a failure of the MaxBotix® Inc. product which may result in personal injury or death. MaxBotix® Inc. will not be held liable for unauthorized use of this component.

About Ultrasonic Sensors

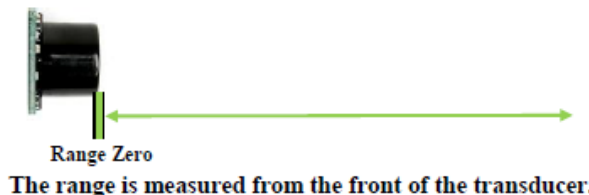
Our ultrasonic sensors are in air, non-contact object detection and ranging sensors that detect objects within an area. These sensors are not affected by the color or other visual characteristics of the detected object. Ultrasonic sensors use high frequency sound to detect and localize objects in a variety of environments. Ultrasonic sensors measure the time of flight for sound that has been transmitted to and reflected back from nearby objects. Based upon the time of flight, the sensor then outputs a range reading.

Pin Out Description

- Pin 1-BW**-*Leave open or hold low for serial output on the TX output. When BW pin is held high the TX output sends a pulse (instead of serial data), suitable for low noise chaining.
- Pin 2-PW**- This pin outputs a pulse width representation of range. The distance can be calculated using the scale factor of 147uS per inch.
- Pin 3-AN**- Outputs analog voltage with a scaling factor of ($V_{cc}/512$) per inch. A supply of 5V yields ~9.8mV/in. and 3.3V yields ~6.4mV/in. The output is buffered and corresponds to the most recent range data.
- Pin 4-RX**- This pin is internally pulled high. The LV-MaxSonar-EZ will continually measure range and output if RX data is left unconnected or held high. If held low the sensor will stop ranging. Bring high for 20uS or more to command a range reading.
- Pin 5-TX**- When the *BW is open or held low, the TX output delivers asynchronous serial with an RS232 format, except voltages are 0-Vcc. The output is an ASCII capital "R", followed by three ASCII character digits representing the range in inches up to a maximum of 255, followed by a carriage return (ASCII 13). The baud rate is 9600, 8 bits, no parity, with one stop bit. Although the voltage of 0-Vcc is outside the RS232 standard, most RS232 devices have sufficient margin to read 0-Vcc serial data. If standard voltage level RS232 is desired, invert, and connect an RS232 converter such as a MAX232. When BW pin is held high the TX output sends a single pulse, suitable for low noise chaining. (no serial data)
- Pin 6+5V- Vcc** - Operates on 2.5V - 5.5V. Recommended current capability of 3mA for 5V, and 2mA for 3V.
- Pin 7-GND**- Return for the DC power supply. GND (& Vcc) must be ripple and noise free for best operation.

Range "0" Location

The LV-MaxSonar-EZ reports the range to distant targets starting from the front of the sensor as shown in the diagram below.



In general, the LV-MaxSonar-EZ will report the range to the leading edge of the closest detectable object. Target detection has been characterized in the sensor beam patterns.

Sensor Minimum Distance

The sensor minimum reported distance is 6-inches (15.2 cm). However, the LV-MaxSonar-EZ will range and report targets to the front sensor face. Large targets closer than 6-inches will typically range as 6-inches.

Sensor Operation from 6-inches to 20-inches

Because of acoustic phase effects in the near field, objects between 6-inches and 20-inches may experience acoustic phase cancellation of the returning waveform resulting in inaccuracies of up to 2-inches. These effects become less prevalent as the target distance increases, and has not been observed past 20-inches.

Appendix D: Arduino Code

MAIN CODE:

```
/******  
*****
```

This code is designed for a wirelessly controlled password based security system with cool features.

Authors: Shruti Motiwale, Dan Givigliano

```
//ACKNOWLEDGEMENTS:
```

Some help for code was taken from the following sources and authors

```
/-----Sonar-----
```

```
/* Mode filter
```

Author: Jason Lessels

Date created: 2011/June/06

Lisence: GPL (=>2)

This work has been compiled using many sources mainly posts/wiki posts from;

Allen, Bruce (2009/July/23) and Gentles, Bill (2010/Nov/12)

```
/-----Sonar-----
```

```
/-----Trellis-----
```

This is a test example for the Adafruit Trellis w/HT16K33

Designed specifically to work with the Adafruit Trellis

----> <https://www.adafruit.com/products/1616>

----> <https://www.adafruit.com/products/1611>

These displays use I2C to communicate, 2 pins are required to interface

Adafruit invests time and resources providing this open source code, please support Adafruit and open-source hardware by purchasing products from Adafruit!

Written by Limor Fried/Ladyada for Adafruit Industries.

MIT license, all text above must be included in any redistribution

```
/-----Trellis-----*/
```

```
//
```

```
/*Keeping track of all digital pins
```

```
LCD = 2
```

```
newUser = 3
```

```
Sonar = 6
```

```
activate pin = 7
```

```
Speakers = 8
```

```
Servo = 9
```

```
shutdown pin = 10
```

```
door unlocked = 11
```

```
red LED = 12
```

white LED = 13

*/

```
/*  
*****  
******/
```

```
// TRELLIS-----Trellis
```

```
#include <Wire.h>
```

```
#include "Adafruit_Trellis.h"
```

```
int user = 0; //stores the username
```

```
//Variables for keeping track of keys pressed for password
```

```
//:-----
```

```
int counter = 0;
```

```
int state = 0;
```

```
int states[] = {1,1,1,1, 1,1,1,1, 1,1,1,1, 1,1,1,1};
```

```
int key = 0;
```

```
int loadingPattern[] = {0, 1, 2, 3, 7, 11, 15, 14, 13, 12, 8, 4, 5, 6, 10, 9};
```

```
//:-----
```

```
Adafruit_Trellis matrix0 = Adafruit_Trellis();
```

```
Adafruit_TrellisSet trellis = Adafruit_TrellisSet(&matrix0);
```

```
#define NUMTRELLIS 1
```

```
#define numKeys (NUMTRELLIS * 16)
```

```
//SERVO-----Servo
```

```
//#include <Servo.h>
```

```
//Servo myservo;
```

```
byte servo = 3;
```

```
//int pos = 110;
```

```
const int servoPin = 9; // the pin number (no PWM) of the servo.
```

```
const long delayTime = 10; // servo movement delay time in millis.
```

```
int myAngle = 120; // angle of the servo (roughly in degrees) 0-180.
```

```
int pulseWidth = 0; // angle to microseconds.
```

```
//Sonar -----Sonar
```

```
const int sonar = 6; //variables needed to store values
```

```
int arraysize = 9; //quantity of values to find the median (sample size). Needs to be an odd number
```

```
//declare an array to store the samples. not necessary to zero the array values here, it just makes the code clearer
```

```
int rangevalue[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0};
```

```
long pulse;
```

```
int modE;
```

```
int count = 0;
```

```
int dist=200;
```

```
// LCD with Motor-----LCD
```

```

#include <SoftwareSerial.h>
// Create a software serial port!
SoftwareSerial lcd = SoftwareSerial(0,2);
int password=0;

// Music-----Music

#include "pitches.h"
int notes[] = { 0,
NOTE_C4, NOTE_CS4, NOTE_D4, NOTE_DS4, NOTE_E4, NOTE_F4, NOTE_FS4, NOTE_G4, NOTE_GS4,
NOTE_A4, NOTE_AS4, NOTE_B4,
NOTE_C5, NOTE_CS5, NOTE_D5, NOTE_DS5, NOTE_E5, NOTE_F5, NOTE_FS5, NOTE_G5, NOTE_GS5,
NOTE_A5, NOTE_AS5, NOTE_B5,
NOTE_C6, NOTE_CS6, NOTE_D6, NOTE_DS6, NOTE_E6, NOTE_F6, NOTE_FS6, NOTE_G6, NOTE_GS6,
NOTE_A6, NOTE_AS6, NOTE_B6,
NOTE_C7, NOTE_CS7, NOTE_D7, NOTE_DS7, NOTE_E7, NOTE_F7, NOTE_FS7, NOTE_G7, NOTE_GS7,
NOTE_A7, NOTE_AS7, NOTE_B7
};
int melody[] = {
  NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4};

int noteDurations[] = {
  4, 8, 8, 4, 4, 4, 4 };

#define melodyPin 8
//Mario main theme melody
int mario[] = {
  NOTE_E7, NOTE_E7, 0, NOTE_E7,
  0, NOTE_C7, NOTE_E7, 0,
  NOTE_G7, 0, 0, 0,
  NOTE_G6, 0, 0, 0, };

/*
NOTE_C7, 0, 0, NOTE_G6,
0, 0, NOTE_E6, 0,
0, NOTE_A6, 0, NOTE_B6,
0, NOTE_AS6, NOTE_A6, 0,

NOTE_G6, NOTE_E7, NOTE_G7,
NOTE_A7, 0, NOTE_F7, NOTE_G7,
0, NOTE_E7, 0, NOTE_C7,
NOTE_D7, NOTE_B6, 0, 0,

NOTE_C7, 0, 0, NOTE_G6,
0, 0, NOTE_E6, 0,
0, NOTE_A6, 0, NOTE_B6,
0, NOTE_AS6, NOTE_A6, 0,

```

```
NOTE_G6, NOTE_E7, NOTE_G7,  
NOTE_A7, 0, NOTE_F7, NOTE_G7,  
0, NOTE_E7, 0, NOTE_C7,  
NOTE_D7, NOTE_B6, 0, 0
```

```
};
```

```
*/
```

```
//Mario main theme tempo
```

```
int tempo[] = {
```

```
12, 12, 12, 12,
```

```
12, 12, 12, 12,
```

```
12, 12, 12, 12,
```

```
12, 12, 12, 12,
```

```
12, 12, 12, 12,
```

```
12, 12, 12, 12,
```

```
12, 12, 12, 12,
```

```
12, 12, 12, 12,
```

```
9, 9, 9,
```

```
12, 12, 12, 12,
```

```
12, 12, 12, 12,
```

```
12, 12, 12, 12,
```

```
12, 12, 12, 12,
```

```
12, 12, 12, 12,
```

```
12, 12, 12, 12,
```

```
12, 12, 12, 12,
```

```
9, 9, 9,
```

```
12, 12, 12, 12,
```

```
12, 12, 12, 12,
```

```
12, 12, 12, 12,
```

```
};
```

```
const int red = 12;
```

```
const int green = 13;
```

```
int z;
```

```
const int shutDown = 10; //Pin to inform mega about system shutdown
```

```
const int unlock = 11; //Pin to inform mega about door unlock
```

```
const int activate = 7; //Pin to get information from Mega to restart the system
```

```
const int newUser = 3; //Pin to inform that a person w/o username is at the door
```

```
//-----Setup Loop
```

```
void setup() {
```

```
Serial.begin(9600);
```

```
Serial.println("TEST");
```

```

pinMode(4, OUTPUT); //buzzer
pinMode(13, OUTPUT); //Green led indicator when singing a note
pinMode(12, OUTPUT); //Red led indicator when singing a note
pinMode(11, OUTPUT); //door unlock pin
pinMode(10, OUTPUT); //system shutdown pin
pinMode(7, INPUT); //restart after shutdown
pinMode(3, OUTPUT); //for new user

Serial.println("Initializing... ");
// Servo Setup-----Servo Setup

//myservo.attach(9); // attaches the servo on pin 9 to the servo object
Serial.println("Motor... ");

//Sonar Setup----- --Sonar Setup
pinMode(sonar, INPUT); // used to read the pulse sent by MaxSonar
//The Pulse Width has a scale factor of 147 uS per Inch.
Serial.println("Sonar... ");

//Trellis Setup-----Trellis Setup
trellis.begin(0x70);
Serial.println("Trellis... ");

//LCD Setup-----LCD Setup

lcd.begin(9600);

// clear screen
lcd.write(0xFE);
lcd.write(0x58); // this will clear the screen of any text
delay(10); // we suggest putting delays after each command

// turns backlight off
lcd.write(0xFE);
lcd.write(0x46);
Serial.println("LCD... ");
}

//-----Void Loop
void loop() {

// Motion Sensor----- Sonar Detection
//Serial.println("Sonar Detection ");
deactivate(); //deactivates everything
Serial.println("Nothing!");
dist = distance(); // pulseIn records pulsewidth which is the time in mircoseconds it takes the sonar input
Pin to go
// from 0 to high. The value is coverted to inches using the scale factor

```

```
// of 147 uS per Inch.
```

```
Serial.println(dist);
```

```
while(dist<=100){ //If distance becomes less than a meter
```

```
  Serial.println("Motion Detected!");
```

```
  digitalWrite(green,HIGH);
```

```
  digitalWrite(red,HIGH);
```

```
  delay(50);
```

```
  digitalWrite(green,LOW);
```

```
  digitalWrite(red,LOW);
```

```
  delay(50);
```

```
  digitalWrite(green,HIGH);
```

```
  digitalWrite(red,HIGH);
```

```
  delay(50);
```

```
  digitalWrite(green,LOW);
```

```
  digitalWrite(red,LOW);
```

```
  delay(50);
```

```
Serial.println(" ");
```

```
delay(25);
```

```
dist = distance();
```

```
Serial.println(dist);
```

```
while(dist<50){ //I think I detected something at half a meter
```

```
  Serial.println("Coming closer...");
```

```
  Serial.println(" ");
```

```
  delay(25);
```

```
dist = distance();
```

```
Serial.println(dist);
```

```
count = 0;
```

```
while(dist<30){ //Looks like that something is at 30 cms now
```

```
  Serial.println("Almost here...");
```

```
  count++; //Check for four seconds if someone is actually waiting at the door
```

```
delay(25);
```

```
dist = distance();
```

```
Serial.println(dist);
```

```
if(count>=2){ //Huh! Looks like someone's at the door!
```

```
  digitalWrite(red,HIGH);
```

```
  count=0;
```

```
  Serial.println("I think someone's at the door");
```

```
  activateLCD();
```

```
  activateTrellis();
```

```
  awayMessage();
```

```
  state = 0;
```

```

int attempts = 0;
while(!state){//runs through the loop till correct password is entered
  askUsername();
  user = enterUsername();
  /*if(user==0) {digitalWrite(newUser, HIGH); }
  state = 1;
  break;
  dist=200;
  delay(10000); //suspend system till the owner opens the door
  */
  hello(user); //say hello to the user with their name
  askPassword();
  enterPassword(user);
  attempts = attempts++;
  if (attempts>=3){ //to keep a check on maximum number of incorrect attempts
    dist = 200;//this makes sure that it comes out of all while loops for distance
    state = 1;//comes out of the current password loop
    //Clear Screen
    lcd.write(0xFE);
    lcd.write(0x58); // this will clear the screen of any text
    delay(10); // we suggest putting delays after each command
    // go 'home'
    lcd.write(0xFE); //Places the cursor at location (1, 1)
    lcd.write(0x48);
    delay(10); // we suggest putting delays after each command
    // Write
    lcd.write(0xFE);
    lcd.write(0xD0);
    lcd.write(0x255);
    lcd.write(0x1);
    lcd.write(0x1);

    lcd.println("System ");
    lcd.println(" Shutdown!!!");
    digitalWrite(shutDown, HIGH);//inform Mega
    delay(500);
    digitalWrite(shutDown, LOW);

    for(z=0; z<3; z++){
      // LCD turns backlight off
      lcd.write(0xFE);
      lcd.write(0x46);
      delay(100);

      // LCD turns backlight on
      lcd.write(0xFE);
      lcd.write(0x42);
      delay(100);
    }
  }
}

```



```

    }
    deactivate();//turns off LCD and trellis to save battery
    if(true) //change true to false to use the other method to suspend
    delay(3600000);//systems gets suspended for 1 hour
    else{
        while(!digitalRead(activate)){ //will stay in this loop until Mega sends a one to this pin
        }
    }
}
}
}
}
}
}
delay(1000); //Take readings once every second

}

//-----

```

FUNCTIONS:

Functions for Password:

```

int enterPassword(int name)
{

    Serial.print("Please Enter Password for user ");
    Serial.println(name);
    if (name==1){
        Password(3,6,9,12);//if password is entered correctly, sets status to 1
    }
    else if (name==2){
        Password(2,3,0,9);//if password is entered correctly, sets status to 1
    }
    else{
        Password(1,7,9,11);//just a dummy function to get the password input
        state=0;
    }
    if(state ==1) {
        Serial.println("Correct Password");
        Serial.println("Unlocking...");
        //-----Printing this msg on LCD-----//
        clrLCD();

        // Write
        lcd.println(" Correct!");
        sing(1);
        delay(1000);
    }
}

```

```

clrLCD();

// Blink on
lcd.write(0xFE);
lcd.write(0x54); //turn off the blinking block cursor

// Write
lcd.println(" UNLOCKING....");
delay(10);

//-----Unlocking the motor-----//
/*int i = 0;
for(pos = 120; pos >=0 ; pos -= 2) // goes from 0 degrees to 180 degrees
{
    // in steps of 1 degree

    loading();
    myservo.write(0);          // sets the servo position according to the scaled value
    myservo.write(pos); // tell servo to go to position in variable 'pos'
    // waits 15ms for the servo to reach the position
    delay(15);
} */
for (myAngle = 120; myAngle >= 0; myAngle-=2) {
    loading();
    digitalWrite(red,LOW);
    servoPulse(servoPin, myAngle);
}

digitalWrite(unlock, HIGH);//inform Mega

//-----//
lcd.write(0xFE); //Places the cursor at location (1, 1)
lcd.write(0x48);
delay(10); // we suggest putting delays after each command

//Blink off
lcd.write(0xFE);
lcd.write(0x54); //turn off the blinking block cursor

// Write
lcd.println(" UNLOCKED!");
digitalWrite(red,LOW);
digitalWrite(green,HIGH);
digitalWrite(unlock, LOW);
delay(2000);

clrLCD();
// Blink on

```

```

lcd.write(0xFE);
lcd.write(0x54); //turn off the blinking block cursor
// Write

lcd.println(" LOCKING...3");
//Play Tone
int noteDuration = 1000/8;
tone(8, notes[10],noteDuration);
int pauseBetweenNotes = noteDuration * 1.30;
delay(pauseBetweenNotes);
noTone(8);
delay(900);
clrLCD();
lcd.println(" LOCKING...2");
//Play Tone
tone(8, notes[10],noteDuration);
delay(pauseBetweenNotes);
noTone(8);
delay(900);
clrLCD();
lcd.println(" LOCKING...1");
//Play Tone
tone(8, notes[10],noteDuration);
delay(pauseBetweenNotes);
noTone(8);
delay(900);

/*for(pos = 0; pos<=120; pos+=2) // goes from 180 degrees to 0 degrees
{
unloading();
myservo.write(pos); // tell servo to go to position in variable 'pos'
delay(15); // waits 15ms for the servo to reach the position
} */
// lock the motor again
for (myAngle = 0; myAngle <= 120; myAngle+=2) {
unloading();
servoPulse(servoPin, myAngle);
}

clrLCD();
// Write
lcd.println(" LOCKED ");
digitalWrite(red,HIGH);
digitalWrite(green,LOW);
//myservo.write(110);
servoPulse(servoPin, 110);
delay(3000);
//-----end of locking and unlocking-----//

```

```

    clrLCD();

    // LCD turns backlight off
    lcd.write(0xFE);
    lcd.write(0x46);

    delay(1000);
}

else if (state == 0){
    Serial.println("Incorrect Password");
    clrLCD();
    // Write
    lcd.println(" Incorrect");
    delay(2000);
}
else Serial.println("Something is still not working");
}

// _____

int Password(int i1, int i2, int i3, int i4){ //passes the keys pressed for password. A separate function for
this makes it easy to add password reset functionality in future
    states[0] = enterKey(i1);
    resetKeys();
    states[1] = enterKey(i2);
    resetKeys();
    states[2] = enterKey(i3);
    resetKeys();
    states[3] = enterKey(i4);
    resetKeys();
    state = 1;
    for (int j=0; j<16; j++){
        state = state*states[j];
    }
}

```

Functions for Trellis:

```

//Trellis Activation
void activateTrellis(){
    Serial.println("Activating Trellis...");

    // light up all the LEDs in order
    for (uint8_t i=0; i<numKeys; i++) {
        trellis.setLED(i);
        //Play Tone
    }
}

```

```

    int noteDuration = 1000/8;
    tone(8, notes[i*2],noteDuration);
    int pauseBetweenNotes = noteDuration;
    delay(pauseBetweenNotes);
    noTone(8);
    trellis.writeDisplay();
    delay(15);

}
// then turn them off
for (uint8_t i=0; i<numKeys; i++) {
    trellis.clrLED(i);
    //Play Tone
    int noteDuration = 1000/8;
    tone(8, notes[32-i*2],noteDuration);
    int pauseBetweenNotes = noteDuration;
    delay(pauseBetweenNotes);
    noTone(8);
    trellis.writeDisplay();
    delay(15);
}
}

//-----
int enterUsername()
{
    int name = 0;
    Serial.println("Please Enter Username.");
    while (!trellis.readSwitches()) { //keeps running through the loop until no key is pressed
        //Serial.println("Please enter a key.");
        delay(30);
    }
    for (uint8_t i=0; i<numKeys; i++) {
        // if it was pressed, turn it on
        if (trellis.justPressed(i)) {
            Serial.print("You pressed "); Serial.println(i);
            trellis.setLED(i);
            //Play Tone
            int noteDuration = 1000/4;
            tone(8, notes[10+i*2],noteDuration);
            int pauseBetweenNotes = noteDuration;
            delay(pauseBetweenNotes);
            noTone(8);
            name = i;
        }
    }
    // tell the trellis to set the LEDs we requested
    trellis.writeDisplay();
}

```

```

    delay(500);
    resetKeys();
    return name;
}

//-----
int enterKey(int button) //function for registering the input from each key and checking for correctness
{
    delay(30); // 30ms delay is required, don't remove me!
    int value = 100;
    // If a button was just pressed or released...
    while (!trellis.readSwitches()) { //keeps running through the loop until no key is pressed
        //Serial.println("Please enter a key.");
        delay(30);
    }
    // go through every button

    for (uint8_t i=0; i<numKeys; i++) {
        // if it was pressed, turn it on
        if (trellis.justPressed(i)) {
            Serial.print("You pressed "); Serial.println(i);
            trellis.setLED(i);
            //Play Tone
            int noteDuration = 1000/4;
            tone(8, notes[10+i*2],noteDuration);
            int pauseBetweenNotes = noteDuration;
            delay(pauseBetweenNotes);
            noTone(8);
            if (i==button) {
                //Serial.println("Correct!");
                value = 1;
            }
            else {
                //Serial.println("Incorrect!");
                value = 0;
            }
        }
    }
    // tell the trellis to set the LEDs we requested
    trellis.writeDisplay();
    return value;
}

//-----
void resetKeys()//function for clearing all LEDs
{
    delay(250);
    while (!trellis.readSwitches()) { //keeps running through the loop until no key is pressed
        delay(30);
    }
}

```

```

}
//Serial.println("Turning keys off");
for (uint8_t i=0; i<numKeys; i++) {
    trellis.clrLED(i);
    trellis.writeDisplay();
}
}

#define INTPIN 5
int i=0;
void loading(){ //function for moving LED display on the keypad while locking

    switch(myAngle){
    case 0:
        i=0;
        trellis.setLED(loadingPattern[0]);
        trellis.writeDisplay();
        break;

    case 8:
        i = 1;
        trellis.setLED(loadingPattern[i]);
        trellis.writeDisplay();
        break;

    case 14:
        i = 2;
        trellis.setLED(loadingPattern[i]);
        trellis.writeDisplay();
        break;

    case 22: i=3;
        trellis.setLED(loadingPattern[i]);
        trellis.writeDisplay();
        break;

    case 28: i = 4;
        trellis.setLED(loadingPattern[i]);
        trellis.writeDisplay();
        break;

    case 36: i = 5;
        trellis.setLED(loadingPattern[i]);
        trellis.writeDisplay();
        break;

    case 42: i = 6;
        trellis.setLED(loadingPattern[i]);

```

```
trellis.writeDisplay();  
break;
```

```
case 50: i = 7;  
trellis.setLED(loadingPattern[i]);  
trellis.writeDisplay();  
break;
```

```
case 56: i = 8;  
trellis.setLED(loadingPattern[i]);  
trellis.writeDisplay();  
break;
```

```
case 64 : i = 9;  
trellis.setLED(loadingPattern[i]);  
trellis.writeDisplay();  
break;
```

```
case 70: i = 10;  
trellis.setLED(loadingPattern[i]);  
trellis.writeDisplay();  
break;
```

```
case 78: i = 11;  
trellis.setLED(loadingPattern[i]);  
trellis.writeDisplay();  
break;
```

```
case 84: i = 12;  
trellis.setLED(loadingPattern[i]);  
trellis.writeDisplay();  
break;
```

```
case 90: i = 13;  
trellis.setLED(loadingPattern[i]);  
trellis.writeDisplay();  
break;
```

```
case 100: i = 14;  
trellis.setLED(loadingPattern[i]);  
trellis.writeDisplay();  
break;
```

```
case 110: i = 15;  
trellis.setLED(loadingPattern[i]);  
trellis.writeDisplay();  
break;
```



```
    case 120: i = 16;
    trellis.setLED(loadingPattern[i]);
    trellis.writeDisplay();
    break;
  }
}
```

```
#define INTPIN 5
```

```
void unloading(){ //function for moving LED display on the keypad while unlocking
```

```
    switch(myAngle){
    case 0:
    i=0;
    trellis.clrLED(loadingPattern[0]);
    trellis.writeDisplay();
    break;
```

```
    case 8:
    i = 1;
    trellis.clrLED(loadingPattern[i]);
    trellis.writeDisplay();
    break;
```

```
    case 14:
    i = 2;
    trellis.clrLED(loadingPattern[i]);
    trellis.writeDisplay();
    break;
```

```
    case 22: i=3;
    trellis.clrLED(loadingPattern[i]);
    trellis.writeDisplay();
    break;
```

```
    case 28: i = 4;
    trellis.clrLED(loadingPattern[i]);
    trellis.writeDisplay();
    break;
```

```
    case 36: i = 5;
    trellis.clrLED(loadingPattern[i]);
    trellis.writeDisplay();
    break;
```

```
    case 42: i = 6;
    trellis.clrLED(loadingPattern[i]);
    trellis.writeDisplay();
    break;
```

```
case 50: i = 7;
trellis.clrLED(loadingPattern[i]);
trellis.writeDisplay();
break;
```

```
case 56: i = 8;
trellis.clrLED(loadingPattern[i]);
trellis.writeDisplay();
break;
```

```
case 64: i = 9;
trellis.clrLED(loadingPattern[i]);
trellis.writeDisplay();
break;
```

```
case 70: i = 10;
trellis.clrLED(loadingPattern[i]);
trellis.writeDisplay();
break;
```

```
case 78: i = 11;
trellis.clrLED(loadingPattern[i]);
trellis.writeDisplay();
break;
```

```
case 84: i = 12;
trellis.clrLED(loadingPattern[i]);
trellis.writeDisplay();
break;
```

```
case 90: i = 13;
trellis.clrLED(loadingPattern[i]);
trellis.writeDisplay();
break;
```

```
case 100: i = 14;
trellis.clrLED(loadingPattern[i]);
trellis.writeDisplay();
break;
```

```
case 110: i = 15;
trellis.clrLED(loadingPattern[i]);
trellis.writeDisplay();
break;
```

```
case 120: i = 16;
trellis.clrLED(loadingPattern[i]);
```

```
trellis.writeDisplay();  
break;  
}
```

```
}
```

Functions for LCD:

//LCD Activation

```
void activateLCD(){  
  Serial.println("Activating LCD... ");  
  //myservo.write(pos);  
  servoPulse(servoPin, myAngle);  
  uint8_t red, green, blue;  
  lcd.write(0xFE);  
  lcd.write(0xD0);  
  lcd.write(0x255);
```

```
// set the size of the display if it isn't 16x2 (you only have to do this once)  
lcd.write(0xFE); //Tells the backpack to watch for a special command next.  
lcd.write(0xD1); // Sets LCD size  
lcd.write(16);   // 16 columns  
lcd.write(2);    // 2 rows  
delay(10);
```

```
// we suggest putting delays after each command to make sure the data  
// is sent and the LCD is updated.  
//set the contrast, 200 is a good place to start, adjust as desired  
lcd.write(0xFE);  
lcd.write(0x50); //set the display contrast.  
lcd.write(200);  // 180-220 works well  
delay(10);
```

```
// set the brightness - we'll max it (255 is max brightness)  
lcd.write(0xFE);  
lcd.write(0x99); // Sets the brightness  
lcd.write(255);  // 0-255  
delay(10);
```

```
// turn off cursors  
lcd.write(0xFE);  
lcd.write(0x4B); //turn off the underline cursor  
lcd.write(0xFE);  
lcd.write(0x54); //turn off the blinking block cursor
```

```

// create a custom character
lcd.write(0xFE);
lcd.write(0x4E);    // Creates a custom character
lcd.write((uint8_t)0); // location #0 (0-7spots)
lcd.write((uint8_t)0x00); // 8 bytes of character data
lcd.write(0x0A);    // New line character or '\n'
lcd.write(0x15);
lcd.write(0x11);
lcd.write(0x11);
lcd.write(0x0A);
lcd.write(0x04);
lcd.write((uint8_t)0x00);
delay(10); // we suggest putting delays after each command

clrLCD();
// Write
lcd.print("Hello!");
// lcd.write((uint8_t)0); // to print the custom character, 'write' the location
delay(500); // we suggest putting delays after each command
}

//LCD Requests Username
void askUsername(){
  clrLCD();
  // Write
  lcd.print(" Please");
  lcd.println(" Enter");
  lcd.print(" Username");
  delay(500);
}

//LCD Requests Password
void askPassword(){
  clrLCD();
  // Write
  lcd.print(" Please");
  lcd.println(" Enter");
  lcd.print(" Password");
  delay(500);
}

//LCD says Hello
int hello(int user){
  switch (user){
    case 1:
      clrLCD();
      // Write

```

```

lcd.print(" Hello");
lcd.println(" Dan!");
Serial.println("Hello Dan!");
delay(500);
break;

case 2:
  clrLCD();
  // Write
  lcd.print(" Hello");
  lcd.println(" Shruti!");
  Serial.println("Hello Shruti!");
  delay(500);
  break;
}
delay(1000);
}

void clrLCD(){

  //Clear Screen
  lcd.write(0xFE);
  lcd.write(0x58); // this will clear the screen of any text
  delay(10); // we suggest putting delays after each command
  // go 'home'
  lcd.write(0xFE); //Places the cursor at location (1, 1)
  lcd.write(0x48);
  delay(10); // we suggest putting delays after each command

}

//Awat Message
int n=0;

void awayMessage(){
  /*
  for( n=0; n<5; n++){
    clrLCD();
    for (int positionCounter = 0; positionCounter < 13; positionCounter++) {
      // scroll one position left:
      lcd.scrollDisplayLeft(" I will be back at Noon Today");
      // wait a bit:
      delay(150);
    }
  }
  */
  //Clear Screen
  lcd.write(0xFE);

```

```

    lcd.write(0x58); // this will clear the screen of any text
    delay(10); // we suggest putting delays after each command
    // go 'home'
    lcd.write(0xFE); //Places the cursor at location (1, 1)
    lcd.write(0x48);
    delay(10); // we suggest putting delays after each command
    // Write
    lcd.print(" Hello Class! ");
    //lcd.println(" be closed today");
    delay(2000);

}

```

Functions for speaker:

```

void sing(int s){
    // iterate over the notes of the melody:
    if(s=1){
        Serial.println(" 'Mario Theme'");
        int size = sizeof(mario) / sizeof(int);
        for (int thisNote = 0; thisNote < size; thisNote++) {

            // to calculate the note duration, take one second
            // divided by the note type.
            //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
            int noteDuration = 1000/tempo[thisNote];

            buzz(melodyPin, mario[thisNote],noteDuration);

            // to distinguish the notes, set a minimum time between them.
            // the note's duration + 30% seems to work well:
            int pauseBetweenNotes = noteDuration * 1.30;
            delay(pauseBetweenNotes);

            // stop the tone playing:
            buzz(melodyPin, 0,noteDuration);

        }
    }
}

void buzz(int targetPin, long frequency, long length) {
    digitalWrite(green,HIGH);
    digitalWrite(red,HIGH);
    long delayValue = 1000000/frequency/2; // calculate the delay value between transitions
    //// 1 second's worth of microseconds, divided by the frequency, then split in half since
    //// there are two phases to each cycle

```

```

long numCycles = frequency * length/ 1000; // calculate the number of cycles for proper timing
//// multiply frequency, which is really cycles per second, by the number of seconds to
//// get the total number of cycles to produce
for (long i=0; i < numCycles; i++){ // for the calculated length of time...
    digitalWrite(targetPin,HIGH); // write the buzzer pin high to push out the diaphragm
    delayMicroseconds(delayValue); // wait for the calculated delay value
    digitalWrite(targetPin,LOW); // write the buzzer pin low to pull back the diaphragm
    delayMicroseconds(delayValue); // wait again or the calculated delay value
}
digitalWrite(green,LOW);
digitalWrite(red,LOW);
}

```

CODE FOR INTERNET ON MEGA:

Code to connect to Pushingbox:

```

////
//
// General code from http://www.pushingbox.com for Arduino + Ethernet Shield (official) v1.2
//
////

#include <SPI.h>
#include <Ethernet.h>

//////////
// MODIFY HERE //
//////////
byte mac[] = { 0x00, 0xAA, 0xBB, 0xCC, 0xDE, 0x19 }; // Mac address of our Ethernet shield

//Developer ID stands for each action
char DEVID1[] = "v97C8DDF06CBE622";//Scenario: Door Unlocked, sends a mail
char DEVID1[] = "vA187A310E63B607";//Scenario: System shutdown, sends a mail, and a notification on
cell phone
//for all the scenarios above, can also can add a url where an image can be uploaded

//Numeric Pin where you the status of each pin is given from arduino uno
uint8_t pinDevid1 = 6; // Door unlocked
uint8_t pinDevid1 = 8; //System Shutdown

// Debug mode
boolean DEBUG = true;
//////////
// End //
//////////

```

```

char serverName[] = "api.pushingbox.com";
boolean pinDevid1State = false;          // Save the last state of the Pin for DEVID1
boolean pinDevid2State = false;          // Save the last state of the Pin for DEVID2
boolean lastConnected = false;           // State of the connection last time through the main loop


// Initialize the Ethernet client library
// with the IP address and port of the server
// that you want to connect to (port 80 is default for HTTP):
EthernetClient client;

void setup() {
  Serial.begin(9600);
  pinMode(pinDevid1, INPUT);

  // start the Ethernet connection:
  if (Ethernet.begin(mac) == 0) {
    Serial.println("Failed to configure Ethernet using DHCP");
    // no point in carrying on, so do nothing forevermore:
    while(true);
  }
  else{
    Serial.println("Ethernet ready");
    // print the Ethernet board/shield's IP address:
    Serial.print("My IP address: ");
    Serial.println(Ethernet.localIP());
  }
  // give the Ethernet shield a second to initialize:
  delay(1000);
}

void loop()
{
  ///
  // Listening for the pinDevid1 state
  ///
  if (digitalRead(pinDevid1) == HIGH && pinDevid1State == false)
  {
    if(DEBUG){Serial.println("pinDevid1 is HIGH");}
    pinDevid1State = true;
    //Sending request to PushingBox when the pin is HIGH
    sendToPushingBox(DEVID1);
    delay(5000); //to avoid multiple requests being sent
  }
  if (digitalRead(pinDevid2) == HIGH && pinDevid2State == false)
  {
    if(DEBUG){Serial.println("pinDevid1 is HIGH");}
    pinDevid2State = true;
  }
}

```



```

    //Sending request to PushingBox when the pin is HIGH
    sendToPushingBox(DEVID2);
    delay(5000);//to avoid multiple requests being sent
}

//DEBUG part
// this write the respons from PushingBox Server.
// You should see a "200 OK"
if (client.available()) {
    char c = client.read();
    if(DEBUG){Serial.print(c);}
}

// if there's no net connection, but there was one last time
// through the loop, then stop the client:
if (!client.connected() && lastConnected) {
    if(DEBUG){Serial.println();}
    if(DEBUG){Serial.println("disconnecting.");}
    client.stop();
}
lastConnected = client.connected();
}

//Function for sending the request to PushingBox
void sendToPushingBox(char devid[]){
    client.stop();
    if(DEBUG){Serial.println("connecting...");}

    if (client.connect(serverName, 80)) {
        if(DEBUG){Serial.println("connected");}

        if(DEBUG){Serial.println("sendind request");}
        client.print("GET /pushingbox?devid=");
        client.print(devid);
        client.println(" HTTP/1.1");
        client.print("Host: ");
        client.println(serverName);
        client.println("User-Agent: Arduino");
        client.println();
    }
    else {
        if(DEBUG){Serial.println("connection failed");}
    }
}

```

Code to connect to Temboo.com:

//Calls the owner on system shutdown and gives him an option to restart it by pressing a key, which will write HIGH on pin6.

```
/* Setup shield-specific #include statements */
#include <SPI.h>
#include <Dhcp.h>
#include <Dns.h>
#include <Ethernet.h>
#include <EthernetClient.h>
#include <Temboo.h>
#include "TembooAccount.h" // Contains Temboo account information
```

```
byte ethernetMACAddress[] = ETHERNET_SHIELD_MAC;
EthernetClient client;
```

```
// The number of times to trigger the action if the condition is met.
// We limit this so you won't use all of your Temboo calls while testing.
int maxCalls = 10;
```

```
// The number of times this Choreo has been run so far in this sketch.
int calls = 0;
```

```
void setup() {
  Serial.begin(9600);

  // For debugging, wait until the serial console is connected.
  delay(4000);
  while(!Serial);

  Serial.print("DHCP:");
  if (Ethernet.begin(ethernetMACAddress) == 0) {
    Serial.println("FAIL");
    while(true);
  }
  Serial.println("OK");
  delay(5000);

  // Initialize pins
  pinMode(8, INPUT);
  pinMode(6, OUTPUT);
  Serial.println("Setup complete.\n");
}
```

```
void loop() {
```

```

int sensorValue = digitalRead(8);

Serial.println("Sensor: " + String(sensorValue));

if (sensorValue == HIGH) {
  if (calls < maxCalls) {
    Serial.println("\nTriggered! Calling /Library/Nexmo/Voice/CaptureTextToSpeechPrompt...");

    String choice = makeNexmoCall();
    if (choice == "1") {
      digitalWrite(6, HIGH);
    }
    calls++;
  } else {
    Serial.println("\nTriggered! Skipping the action to save Temboo calls during testing.");
    Serial.println("You can adjust or remove the calls/maxCalls if() statement to change this behavior.\n");
  }
}
delay(250);
}

String makeNexmoCall() {
  String choice = "";
  TembooChoreo CaptureTextToSpeechPromptChoreo(client);

  // Set Temboo account credentials
  CaptureTextToSpeechPromptChoreo.setAccountName(TEMBOO_ACCOUNT);
  CaptureTextToSpeechPromptChoreo.setAppKeyName(TEMBOO_APP_KEY_NAME);
  CaptureTextToSpeechPromptChoreo.setAppKey(TEMBOO_APP_KEY);

  // Set profile to use for execution
  CaptureTextToSpeechPromptChoreo.setProfile("arduinoMotionPhone");

  // Set Choreo inputs
  String MaxDigitsValue = "1";
  CaptureTextToSpeechPromptChoreo.addInput("MaxDigits", MaxDigitsValue);
  String ByeTextValue = "Ok, your wish is my command. Goodbye!";
  CaptureTextToSpeechPromptChoreo.addInput("ByeText", ByeTextValue);

  // Set Choreo output filters
  CaptureTextToSpeechPromptChoreo.addOutputFilter("choice", "/digits", "CallbackData");

  // Identify the Choreo to run

  CaptureTextToSpeechPromptChoreo.setChoreo("/Library/Nexmo/Voice/CaptureTextToSpeechPrompt")
;

  // Run the Choreo

```

```

unsigned int returnCode = CaptureTextToSpeechPromptChoreo.run();

// A return code of zero means everything worked
if (returnCode == 0) {
    Serial.println("Done!\n");
} else {
    // A non-zero return code means there was an error
    // Read and print the error message
    while (CaptureTextToSpeechPromptChoreo.available()) {
        char c = CaptureTextToSpeechPromptChoreo.read();
        Serial.print(c);
    }
    Serial.println();
}

// Parse the results
while(CaptureTextToSpeechPromptChoreo.available()) {
    // Read the name of the next output item
    String name = CaptureTextToSpeechPromptChoreo.readStringUntil('\x1F');
    name.trim(); // Use "trim" to get rid of newlines

    // Read the value of the next output item
    String data = CaptureTextToSpeechPromptChoreo.readStringUntil('\x1E');
    data.trim(); // Use "trim" to get rid of newlines

    // Parse the user's choice out of the response data
    if (name == "choice") {
        choice = data;
    }
}

CaptureTextToSpeechPromptChoreo.close();

// Return the choice that the user made
return choice;
}

```

Header File:

```

/*
IMPORTANT NOTE about TembooAccount.h
TembooAccount.h contains your Temboo account information and must be included
alongside your sketch.
*/
#define TEMBOO_ACCOUNT "shrutim" // Your Temboo account name
#define TEMBOO_APP_KEY_NAME "myFirstApp" // Your Temboo app key name
#define TEMBOO_APP_KEY "c1625eae115c4506866b8a2f24dcc84" // Your Temboo app key

```

```
#define ETHERNET_SHIELD_MAC {0x90, 0xA2, 0xDA, 0x0D, 0x4E, 0x77}
```



Home Automation

ME 445 Final Project

ABSTRACT

When thinking about ideas for this project we thought it would be awesome to have home automation like that of JARVIS from the Iron Man movies. We looked into the current types of home automation and many need some sort of action on the part of the user. Our idea was to have a passive home automation system that would respond to the user rather than require commands or directions. The final demo will be of how the house will act when the user wakes up in two different scenarios.

Matt Harkenreader, James
Van Sant

Table of Contents

I.	Introduction	2
II.	Project Components	2
	Description of Components	2
	Bill of Materials	4
	Software Components	4
III.	Component Interaction.....	4
IV.	Communication Flow Chart	5
V.	Problems and Troubleshooting.....	6
	Hardware Issues.....	6
	Software Issues	6
VI.	Closing Thoughts	8
	Lessons Learned	8
	Conclusion.....	8
VII.	References	9
VIII.	Appendices.....	10
	A. Arduino Fio Code.....	10
	B. RedBot Mainboard Code.....	12
	C. Processing Code	18
	D. Data Sheets	25

I. Introduction

The project we chose is a working prototype of an improved and personalized automated home. The idea was born when looking at current options available for home automation. The consumer choices are limited to either heavy dependence on motion sensors and cameras¹ or to connect the home to the internet through the homeowner's phone or cable provider.² These two options have unfortunate drawbacks; motion sensors and cameras are clunky and conspicuous while connecting an entire home to the internet could leave it susceptible to malware.³

To combat these setbacks, our concept focuses around connecting the home to the person rather than devices or the internet. The homeowner would wear a bracelet that contains electronics which communicate with small devices (in our case, Arduinos) installed throughout the home. The devices would then perform different actions depending on the person's location in the house, time of day, and programming. The example portrayed in our demonstration is a common household occurrence: a middle-of-the-night bathroom trip. The bracelet would inform the bedroom-controlling Arduino that the user has left the bed. The Arduino, which knows that it is still nighttime, would turn on the lights to a low level to avoid blinding the user while providing enough light to navigate the room without bumping shins or stubbing toes. Later, when the user wakes up for work, the Arduino now realizes it is later in the morning and will turn the lights on full and display a message about the current weather on the bedroom television screen or computer monitor.

The hardware components include a Sparkfun RedBot mainboard, Arduino Fio microcontroller board, an accelerometer MPU 6050, a strip of Adafruit NeoPixels, and XBee wireless communicator chips. The software components involve two Arduino sketches and one Processing sketch.

II. Project Components

Description of Components

The **Arduino Fio**⁴ is a microcontroller board mainly designed for wireless applications. It has fourteen digital I/O pins (6 PWM), eight analog inputs, a reset button, a resonator, and holes for mounting pin headers. The Fio runs on 3.3V and has a connector for a lithium polymer battery (it can also be powered via a micro USB port).

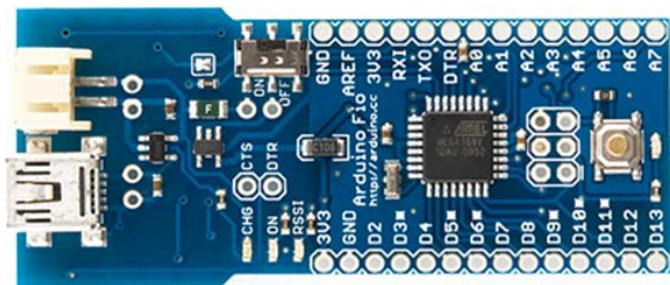


Figure 1: Arduino Fio <http://arduino.cc/en/Main/ArduinoBoardFio>

Its primary feature is an XBee socket on the reverse side (not pictured). This enables the Fio to communicate wirelessly with other XBee-compatible boards provided by Arduino and Sparkfun. In our project, XBees are used so the Fio and RedBot main board are able to communicate with each other – one that runs the accelerometer and one that runs the NeoPixels.



The **XBee⁵** is a wireless communication device that operates within the ISM 2.4GHz frequency. The XBee is good for indoor (30m) and outdoor use (90m line of sight) as well as being very power efficient. It is used to wirelessly communicate between two devices, in our case an Arduino Fio and a RedBot mainboard.

These chips communicate with the serial port so it will be prudent to ensure that the communication between the XBees does not interfere with the communication between the Redbot and Processing.

Figure 2: XBee S1
<http://examples.digi.com/get-started/basic-xbee-802-15-4-chat/>

The **RedBot mainboard⁶** is a robotic platform that works with the Arduino IDE. It uses the same software that the Arduino needs; however, to connect to this board a USB mini-B is needed. This board is a combination motor driver and Arduino which means that shields are not needed (unlike the Arduino). Additionally there is an XBee port built into the chip.

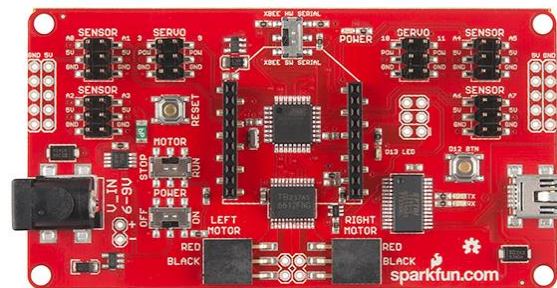


Figure 3: RedBot
<https://www.sparkfun.com/products/12097>



The **Adafruit NeoPixel Digital RGB LED Strip⁷** is a programmable series of LED lights with either a two or three pin JST SM connector on each end (if a full four meter is purchased). The lights are encased in a white-colored flex PCB and are individually addressable, which allows each light to be controlled with 8-bit PWM precision and needs at least an 8 MHz processor for programming. A power supply that can provide at least 2A of current at 5V is required.

For the project, the NeoPixel will serve two purposes. It will represent bedroom lights for the

Figure 4: NeoPixel strip
<http://www.adafruit.com/product/1138>

2AM portion of the demonstration. Once the accelerometer is tripped, the lights will come on white with a dim brightness to simulate low lighting during the night when the user needs to use the bathroom. When the accelerometer is tripped for the 7AM portion of the demonstration, the lights will display a pattern that corresponds to the current weather conditions outside, which are read from the internet.

The **MPU 6050 Accelerometer**⁸ is a six degree of freedom, three axis gyroscope, and three axis accelerometer. This allows the chip to simultaneously capture data in the x, y, and z directions. It has 16-bit analog to digital conversion hardware for each channel. It uses an I2C-bus to interface with the Arduino/RedBot main board.



Figure 5: MPU 6050
<http://playground.arduino.cc/Main/MPU-6050>

Bill of Materials

The total cost of this project (starting from scratch) would be \$153.24. We were able to find everything we needed in the ME 445 Lab and were able to save money.

Table 1: BOM to complete our project

Item	Model #	Vendor	Retail/Unit Cost	Qty.	Supplier	Cost
Arduino Fio	DEV-10116	Sparkfun	\$24.95	1	ME 445 Lab	\$0.00
Xbee S1	WRL-08665	Sparkfun	\$24.95	2	ME 445 Lab	\$0.00
Neopixel LED Strip	1138	Adafruit	\$24.95/Meter	1	ME 445 Lab	\$0.00
Accelerometer	MPU 6050	gearbest	\$3.49	1	ME 445 Lab	\$0.00
Red Bot main board	ROB-12097	Sparkfun	\$49.95	1	ME 445 Lab	\$0.00
Total			\$153.24			\$0.00

Software Components

Two programming languages were utilized for this project. The first is **Arduino IDE**⁹, a language similar to C that has been the basis for the entire class. Two Arduino sketches are used to program each Fio; one sketch will monitor the accelerometer and send a message once a threshold has been reached, while the other will receive the message from the accelerometer, control the NeoPixel strip, and communicate with the computer.

Processing¹⁰ is the other language used and is primarily a tool for creating graphics on screen. For the purposes of the project, Processing will collect local weather data from the National Weather Service's website and send the data to the NeoPixel Arduino. The Fio will then dictate the pattern of the NeoPixel based upon the weather and also send a message back to Processing. Upon receipt of this message Processing will display a message on the computer screen offering a greeting to the user (as if they just woke up), the current weather conditions, and a suggestion based upon the weather.

III. Component Interaction

The accelerometer will be the first component to do anything. It will sense when the arm of the user has moved into a vertical (downward) position. The accelerometer, once tripped, will signal the Fio to send information over to the Redbot. The Redbot will then do one of two things, depending on the timing of the Fio.

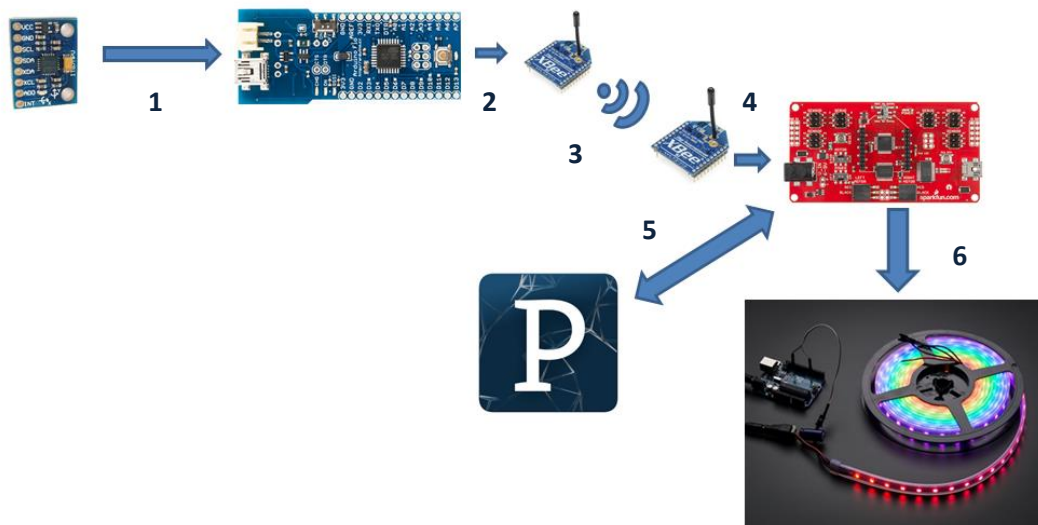
If the accelerometer trips the Fio early in the run of the program, the Redbot will simply turn the lights on dim. This is to simulate somebody waking up in the middle of the night who does not want to be blinded or awake a significant other. After a few seconds, the lights will turn off and the code will advance to the next set of commands which will be executed the next time the Fio is tripped.

After the Fio is tripped a second time by the accelerometer (or a timing threshold has been passed), the Redbot will begin checking for a byte from Processing. The Processing code, which has been running continuously since the beginning, is linked to an RSS feed from the National Weather Service to get real-time local weather data. This data is then analyzed and a custom message for the computer monitor is generated (but not displayed) depending on the current conditions. Additionally, a byte with a specific value for that weather condition is created and sent to the Redbot. Once the Redbot receives the byte it executes a NeoPixel function which imitates the current weather. Then a message is sent back to Processing, which prompts the custom message to be visually displayed on the screen. This entire process simulates the user waking up for work in the morning. The automated home would provide the user with weather information and a suggestion on their bedroom television screen or computer monitor after it sensed that the homeowner has awoken. Their lights would also automatically come on at full brightness (or, as in our case, depict the weather conditions).

For further details on the actual execution of these processes, refer to Appendices A, B, and C for a copy of the code.

IV. Communication Flow Chart

The components communicate in the following manner:



1. The MPU6050 sends raw accelerometer data to the Fio, which translates the information to a -100 – 100 scale.
2. The data is then sent to the XBee for over-the-air communication
3. The XBee transmits the data wirelessly to the second XBee.
4. The second XBee sends the data to the Redbot.
5. Based upon the timing of the XBee data, the Redbot will communicate with Processing and wait for a response before proceeding with its programming
6. The Redbot will send commands to the NeoPixel based upon the message from Processing to create a custom light pattern.

V. Problems and Troubleshooting

With a project that involved so many different components, several issues occurred while working on this project. They were all able to get resolved by the completion of the project.

Hardware Issues

There were three primary problems with the hardware. The first was proper wiring with the accelerometer; we had previously worked with the accelerometer with a lab but had not wired it into the Fio before. After writing a test sketch to read the accelerometer, we tested each analog pin on the Fio to find which ones it needed to be wired into. We eventually discovered that the yellow and blue wires needed to be hooked into pins A4 and A5, respectively.

The second issue came with the XBee communication. The project required the Redbot mainboard to communicate with both the Fio wirelessly through the XBee and with Processing through the serial port. Using an XBee on the Redbot is normally done in hardware serial mode, but this occupies the serial port and prevents Processing from communicating with the Redbot. The solution was to use software serial mode, which generates a virtual serial port on pins A0 and A1. The software serial mode required us to add a new library to the Arduino programming, but it freed up the serial port for communications with Processing.

The only hardware issue we ran into with the NeoPixels was that the wires broke at the solder joint. This was a small issue only and was resolved by stripping the wire, stress relieving the wires by putting a loop in them, and then soldering the wires back to their respective positions.

Software Issues

There were several challenges in coding that needed to be addressed. One of the major issues was the fact that we were programming in two separate languages and they needed to communicate with each other. Additionally, the fact that we had to learn Processing as we worked further complicated matters.

Processing. The first issue that occurred was extracting weather data from the internet, which is done by reading an RSS feed. An RSS, which is published in an XML document, is made up of “parents” and “children.” Items from the RSS that are children can only be accessed by first calling the parent and then calling the child, but this can only be done one generation at a time. For example, consider an example of an RSS feed shown below:

```

- <Organization>
  - <OrganizationDescription>
    <OrganizationIdentifier>TEST_ORG</OrganizationIdentifier>
    <OrganizationFormalName>ABC
      Organization</OrganizationFormalName>
    <OrganizationDescriptionText>Here is a description of the
      organization.</OrganizationDescriptionText>
    <TribalCode>001</TribalCode>
  </OrganizationDescription>
  + <ElectronicAddress>
  + <Telephonic>
  + <OrganizationAddress>
  + <Project>
  + <MonitoringLocation>
  + <MonitoringLocation>
  - <Activity>
    - <ActivityDescription>
      <ActivityIdentifier>10001</ActivityIdentifier>
      <ActivityTypeCode>Field Msr/Obs-Portable Data
        Logger</ActivityTypeCode>
      <ActivityMediaName>Biological Tissue</ActivityMediaName>
      <ActivityMediaSubdivisionName>Surface
        soil/sediment</ActivityMediaSubdivisionName>
      <ActivityStartDate>2006-05-13</ActivityStartDate>
    - <ActivityStartTime>
      <Time>14:20:00</Time>
      <TimeZoneCode>EST</TimeZoneCode>

```

Figure 6: <http://www.mahugh.com/images/blog/2007/05/19/samplexml.jpg>

If we wanted to access the item, “OrganizationFormalName,” we could not access it through the parent, “Organization.” Instead, we would have to access, “Organization,” then call “OrganizationDescription,” and *finally* call “OrganizationFormalName.” This was something we did not initially recognize when calling the weather RSS data and resulted in errors from Processing. Once this was discovered we were able to extract the data we needed.

The second issue with Processing arose when trying to get it to communicate with the Redbot. When running initial tests, it was found that the message that the Redbot received was not the same message that Processing was sending because of an ASCII translation. This problem was resolved by changing the type of variable that was being sent from an integer (int) to a byte, which retains its value during communication. After this change was made, it was discovered that the code would execute as expected, but not consistently or reliably. The problem was found to be in the first draft of the sketch, where a loop was designed to continuously write a message to the serial port until the Redbot responded, at which point it would continue with the rest of its code. This caused some timing issues between the two programs, so the loop was removed to resolve the problems.

Arduino IDE. There were not as many issues writing sketches for the Fio and Redbot as there were with Processing. The major issue, which was addressed above, was with the communication between the two programs. Other than that, the only communication setbacks that we encountered were managing the XBees with Processing. This involved downloading and installing the Redbot SoftwareSerial library and learning how to program with it. Once this was taken care of all communication ran without any faults.

The **NeoPixels** were a challenge in their own right. There were a few examples that came with the library download. Everything we did was extrapolated from the example code(s) provided.

When writing our code for the lightning and wind functions there was no code to reference. The code was made based on trial and error of what we thought would work. The final code took a few iterations to get. The main issue with the NeoPixels was that their reference (the Uberguide) is only example code for particular instances. The Arduino reference is set up much better, it would have helped immensely if Adafruit had a reference like the Arduino reference.

VI. Closing Thoughts

This project was a great experience because it forced us to dive into areas in which we had no prior knowledge. Unlike other assignments that ask students to build upon material they learned in class, this one asked for that and challenged us to try something new.

Lessons Learned

There were many things that we learned while completing this project. Primarily, we were shown that there is not a reason to be apprehensive when asked to learn material independently. Tools such as the Internet are great resources to discover more about what is trying to be accomplished and, perhaps more importantly, that there is a good chance it has been done before. By fearlessly facing a situation and taking the time to understand all of the seemingly inconsequential pieces of the problem, completing the task not only becomes easier but much more can be accomplished than was originally even considered.

Additionally, we learned that there is value in taking the time to weigh simplicity against elegance when programming. There were several occasions where the code was designed to make the computer do less work, but it ended up causing bugs in the programming. By simplifying the code we learned that placing more burdens on the computer is a negligible cost if the program runs as intended.

Conclusion

This project was a textbook exercise in the power of creativity and an idea. The concept, which in its infancy was a set of gloves and kneepads that would alert someone who is blind that they would run into an obstacle, ultimately became a rough alpha prototype for a product which could potentially turn a home into a personal assistant. In addition to the power of creativity, the project showed what could be done when a team is challenged to push their boundaries; each time we approached Mike with a revision to our concept he would often ask what else we could do to make the project more impressive. Without this constant push to do more, our project would not be at the point it is today.

The end result of both our own creative thought processes and the outside forces inspiring a larger vision for the project was a polished prototype that does everything we intended it to do and more. At this point, there is even a chance that this concept can be pursued as a real business venture, and it would not have been possible if we were not pushed to work out the extra bugs, burn the midnight oil, and offered this opportunity to get out of our comfort zone and attempt something entirely new.

VII. References

1. "Home Automation Superstore." Smarthome. N.p., n.d. Web. 16 Dec. 2014. <<http://www.smarthome.com/>>.
2. "The Advantages of Home Automation." *Comcast.com*. N.p., n.d. Web. 16 Dec. 2014. <<http://www.comcast.com/resources/home-automation.html>>.
3. Hill, Kashmir. "When 'Smart Homes' Get Hacked: I Haunted A Complete Stranger's House Via The Internet." *Forbes*. Forbes Magazine, 26 July 2013. Web. 16 Dec. 2014. <<http://www.forbes.com/sites/kashmirhill/2013/07/26/smart-homes-hack/>>.
4. "Arduino - ArduinoBoardFio." *Arduino*. N.p., n.d. Web. 16 Dec. 2014. <<http://arduino.cc/en/Main/ArduinoBoardFio>>.
5. "XBee 1mW Trace Antenna - Series 1 (802.15.4)." - *WRL-11215*. N.p., n.d. Web. 10 Dec. 2014.
6. "RedBot Mainboard." *SparkFun*. N.p., n.d. Web. 16 Dec. 2014. <<https://www.sparkfun.com/products/12097>>.
7. Burgess, Phillip. "NeoPixel Uberguide." (n.d.): n. pag. *Adafruit*. Web. <<http://learn.adafruit.com/downloads/pdf/adafruit-neopixel-uberguide.pdf>>.
8. "MPU-6000/6050 Six-Axis (Gyro + Accelerometer) MEMS MotionTracking™ Devices for Smart Phones, Tablets, and Wearable Sensors." *MEMS Gyro-Accel*. N.p., n.d. Web. 16 Dec. 2014. <<http://invensense.com/mems/gyro/mpu6050.html>>.
9. "Arduino - Software." *Arduino*. N.p., n.d. Web. 16 Dec. 2014. <<http://arduino.cc/en/Main/Software>>.
10. "Processing." *Processing.org*. N.p., n.d. Web. 16 Dec. 2014. <<https://processing.org/>>.

VIII. Appendices

A. Arduino Fio Code

```
// FINAL PROJECT Arduino Fio code (on user arm)
// This sketch is for the ACCELEROMETER FIO
// Created by: Matt Harkenreader, James Van Sant 12/1/14 21:42
// Last Edit: 12/12, 21:55 (MSH)
// Note: Jeff Rowberg contributed to the accelerometer
// portions of this sketch
// I2Cdev and MPU6050 must be installed as libraries, or else the
.cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"
#include "MPU6050.h"
// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE
implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
MPU6050 accelgyro;
int16_t ax, ay, az;
int16_t gx, gy, gz;

byte trip = 1; // Data to send to Redbot

void setup(){
    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

    Serial.begin(9600);
    Serial.flush(); // Empty the Serial Port
    accelgyro.initialize();
}

void loop(){
    int standby = 0;
    while(standby < 1){
        accelgyro.getAcceleration(&ax, &ay, &az);
        long AX = map(ax, -32768, 32768, -100, 100);
        //Serial.println(AX);
        if(AX > 40 || AX < -40){
            Serial.println(trip);
            delay(10);
            standby = 1;
        }
    }
}
```



```
    }  
    else{  
        standby = 0;  
        delay(10); // Wait 10ms between readings  
    }  
}  
}
```

B. RedBot Mainboard Code

```
// FINAL PROJECT: Sketch
// NEOPIXEL SPARKFUN REDBOT
// Created by: Matt Harkenreader, James Van Sant 12/3/14 14:00
// Last Edit: 12/13, 12:17 (JDV)
//NOTE: the functions Slow and Fast are borrowed code from the example
theatrechase found in the NeoPixel sample code
//Credit to NeoPixel, No single coder found.
//Adafruit library needed to run NeoPixels
#include <Adafruit_NeoPixel.h>
#include <RedBot.h> //Redbot SoftwareSerial library needed to run xBee
with Processing
#include <RedBotSoftwareSerial.h> //Define a new serial port for the
XBee on pins A0 and A1
RedBotSoftwareSerial xBee;
#define PIN 14 // defines pin 14 as the pin that the NeoPixels (NP)
will be plugged into
Adafruit_NeoPixel strip = Adafruit_NeoPixel(48, PIN, NEO_GRB +
NEO_KHZ800); // Sets the number, pin, color code, and hz of NP

int conditions; // Weather message received from Processing
int temp; // Temperature received from Processing
int signal = 13; // Use LED on pin 13 for code status
byte w; // Message to read from Processing
byte clean; //Variable used to "scrub" the xBee port until it's empty
int stat; // Integer to bypass while loop
int counter; // Artificial clock to differentiate between 2AM and 7AM
void setup(){
  pinMode(signal, OUTPUT); // Make pin 13 an output
  digitalWrite(signal, LOW); // Ensure that the LED is off
  Serial.begin(9600); // Initiate serial communications
  xBee.begin(9600);

  // Empty the serial ports
  Serial.flush();
  xBee.flush();
  strip.begin(); //initialize strip
  strip.show(); //turns all pixels off on NP
  strip.setBrightness(100); //Set brightness
}
void loop(){ // Set up the standby variables for Processing
  int standby = 0; // Define a variable to wait for serial read
  if(stat > 0){
    standby = 1;
  }
  // BEGIN 2AM processes
  while(counter < 151){ // This code simulates the passage of time
from 2AM to 7AM for the purposes of this demonstration. If the real
time was being monitored, the programming would take advantage of
Processing's timekeeping functions.
    if(xBee.available() > 0){
      digitalWrite(signal, HIGH);
```

```

    for(int i=0; i<48; i++){          //2am code
        strip.setPixelColor(i,127,127,127); //white light
        strip.setBrightness(35);          //set low brightness
        strip.show();
    }
    delay(5000);
    for(int i=0; i<48; i++){          //loop to shut NeoPixels off
        strip.setPixelColor(i,0,0,0);      //turns each pixel off
        strip.show();                    //initialize strip
    }
    break;
}
else{
    digitalWrite(signal, LOW);
    delay(1000); // Wait 1s before next reading
    counter++;
}
}
counter = 200;
// END 2AM processes, BEGIN 7AM processes
while(xBee.available() > 0){
    clean = xBee.read(); //Ensure that the xBee port is empty to avoid
false reading
}
while(standby < 1){
    if(xBee.available() > 0){          // Check to see if data is available
        digitalWrite(signal, LOW);    // Turn off the LED if on
        stat = 1;
        w = Serial.read();
        Serial.write(1);
        standby = 1;
    }
    else{
        digitalWrite(signal, HIGH);
        delay(500); // Make the light blink at half-second intervals
        digitalWrite(signal, LOW);
        delay(500);
        standby = 0;
    }
}
if(w==10 || w==15){ // weather patterns found via Processing
    Sunny(); //Arduino will read to find and compare the numbers
} //To the if statements, the number matches a NeoPixel function
if(w==80 || w==81){
    Windy();
}
if(w==21 || w==22){
    Partly_Cloudy();
}
if(w==23){
    Partly_Sunny();
}

```

```

    if(w==20 || w==70){
        Overcast();
    }
    if(w==32 || w==34){
        Drizzle();
    }
    if(w==30 || w==31 || w==33 || w==35){
        Rain();
    }
    if(w==40 || w==41 || w==42){
        Tstorm();
    }
    if(w==90 || w==91 || w==92 || w==93 || w==94 || w==95 || w==96){
        Wintery_Mix();
    }
    if(w==51 || w==54){
        Flurries();
    }
    if(w==50 || w==52 || w==53 || w==55 || w==56){
        Snow();
    }
    if(w==60 || w==61 || w==62 || w==63 || w==64 || w==65 || w==66){
        Ice();
    }
}

void Sunny(){
    for(int i=0; i<48; i++){ //loop makes whole strip light up
        strip.setPixelColor(i,233,238,26); // produces yellow light
        strip.setBrightness(64); //set brightness of strip
        strip.show();
    }
}

void Partly_Cloudy(){
    for(int i=0; i<48; i++){ //loop makes sections of strip light up
        strip.setPixelColor(i,233,238,26); //colors set to produce yellow
        strip.setBrightness(85); //set brightness of strip
        strip.show(); //initiate command
        i= i+2; //used to skip 2 pixels
    }
    for(int i=1; i<48; i++){ //loop makes sections of strip light up
        strip.setPixelColor(i,127,127,127); //colors set to produce white
        strip.setBrightness(64); //set brightness of strip
        strip.show(); //start of show
        i=i+2; //used to skip 2 pixels
    }
    for(int i=2; i<48; i++){ //loop makes sections of strip light up
        strip.setPixelColor(i,233,238,26); //colors set to produce yellow
        strip.setBrightness(85); //set brightness of strip
        strip.show(); //initiate command
        i=i+2; //used to skip 2 pixels
    }
}

```

```

void Partly_Sunny(){
    for(int i=0; i<48; i++){ //loop makes sections of strip light up
        strip.setPixelColor(i,127,127,127); //colors set to produce yellow
        strip.setBrightness(85); //set brightness of strip
        strip.show(); //initiate command
        i=i+2; //used to skip 2 pixels
    }
    for(int i=1; i<48; i++){ //loop makes sections of strip light up
        strip.setPixelColor(i,233,238,26); //colors set to produce white
        strip.setBrightness(64); //set brightness of strip
        strip.show(); //initiate command
        i=i+2; //used to skip 2 pixels
    }
    for(int i=2; i<48; i++){ //loop makes sections of strip light up
        strip.setPixelColor(i,127,127,127); //colors set to produce yellow
        strip.setBrightness(85); //set brightness of strip
        strip.show(); //initiate command
        i=i+2; //used to skip 2 pixels
    }
}

void Windy(){
    for(int i=0; i<48; i=i+2){ //Uses a gray streak to simulate wind
        strip.setPixelColor(i,154,154,154); //set pixel color to gray
        strip.setBrightness(110); //set brightness
        strip.show(); //initiate command
        delay(25); //delay 25milliseconds
        strip.setPixelColor(i,0,0,0); //shut pixel off
        strip.show(); //initiate command
    }
}

void Overcast(){
    for(int i=0; i<48; i++){ //loop makes whole strip light up
        strip.setPixelColor(i,157,157,157); //colors set to produce
white light
        strip.setBrightness(50); //set brightness of strip DIM
        strip.show(); //initiate command
    }
}

void Drizzle(){
    Slow(strip.Color(0,0,255), 48); //Use Slow function to have blue
light trickle down strip
}

void Rain(){
    Fast(strip.Color(0, 0, 255), 48); //Use Fast funciton to have blue
light shower down strip
}

void Tstorm(){ //Same as rain but introduces a yellow
streak every run
    for(int z=0; z<4; z++){
        Fast(strip.Color(0, 0, 255), 48);
        if (z==0 || z==1 || z==2){
            for(int i=0; i<48; i=i+2){

```

```

        strip.setPixelColor(i,233,238,26);
        strip.setBrightness(110);
        strip.show();
        delay(7);
        strip.setPixelColor(i,0,0,0);
        strip.show();
    }
}

}

void Wintery_Mix(){ //Code used to "mix" rain and snow
for(int c=0; c<11; c++){
    if(c==0 || c==2 || c==4 || c==6 || c==8 || c==10){
        Mix(strip.Color(127, 127,127), 48); // white for snow
    }
    else{
        Mix(strip.Color(0, 0, 255), 48); // blue for rain
    }
}
}

void Flurries(){
    Slow(strip.Color(127, 127, 127), 48);
}

void Snow(){
    Fast(strip.Color(127, 127, 127), 75);
}

void Ice(){
    Slow(strip.Color(47,255,255),48);
}

void Fast(uint32_t c, uint8_t wait) {
    for (int j=0; j<40; j++) { //do 40 cycles of chasing
        for (int q=0; q < 3; q++) {
            for (int i=0; i < strip.numPixels(); i=i+3) {
                strip.setPixelColor(i+q, c); //turn every third pixel on
            }
            strip.show();
            delay(wait);
            for (int i=0; i < strip.numPixels(); i=i+3) {
                strip.setPixelColor(i+q, 0); //turn every third pixel off
            }
        }
    }
}

void Slow(uint32_t c, uint8_t wait) { //this function to be used for
flurries or light rain
    for (int j=0; j<25; j++) { //do 25 cycles of chasing
        for (int q=0; q < 3; q++) {
            for (int i=0; i < strip.numPixels(); i=i+3) {
                strip.setPixelColor(i+q, c); //turn every third pixel on
            }
            strip.show();
            delay(200);
        }
    }
}

```

```

        for (int i=0; i < strip.numPixels(); i=i+3) {
            strip.setPixelColor(i+q, 0);    //turn every third pixel off
        }
    }
}

void Mix(uint32_t c, uint8_t wait) {
    for (int j=0; j<8; j++) { //do 10 cycles of chasing
        for (int q=0; q < 3; q++) {
            for (int i=0; i < strip.numPixels(); i=i+3) {
                strip.setPixelColor(i+q, c);    //turn every third pixel on
            }
            strip.show();

            delay(wait);

            for (int i=0; i < strip.numPixels(); i=i+3) {
                strip.setPixelColor(i+q, 0);    //turn every third pixel off
            }
        }
    }
}

```

C. Processing Code

```
// PROCESSING WEATHER DATA SKETCH
// This sketch will download real-time weather data from an RSS feed
// and generate a message on the computer monitor based upon the time
// of day
XML weather; // Create an XML item to read incoming RSS feed
import processing.serial.*; // Initiate Serial library
Serial com; // Create object from Serial class

PFont font; // Create an object from the font class

String feed = "http://w1.weather.gov/xml/current_obs/KUNV.rss"; // RSS
feed from the National Weather Service for State College
int j = 0; // Counter for conditions loop
int i; // Counter for temp loop
int temp; // Variable for temperature
char temp1; // First temperature digit
char temp2; // Second temperature digit
byte message; // Data to send to the Arduino
String conditions; // Weather in State College
String digit1 = "123456789"; // Comparison string to grab first
temperature digit from RSS feed
String digit2 = "0123456789"; // Comparison string to grab second
temperature digit from RSS feed
String suggestion; // Suggestion to display on the monitor
String rec; // Recommendation based on the weather
String month; // Current month
String day; // Current day
String year; // Current year

void setup(){
  // Size the window, build first background
  size(1920, 1080);
  colorMode(HSB);
  background(160,0,0);

  // Create the font
  font = createFont("Georgia", 32);
  textFont(font);

  // Open the Serial port
  String Port = "COM13";
  com = new Serial(this, Port, 9600);
  com.clear();

  // Pull desired information from the RSS feed
  weather = loadXML(feed);
  XML channel = weather.getChild("channel");
  XML title = channel.getChild("title");
  XML item = channel.getChild("item");
  XML temp = item.getChild("title");
```



```

conditions = temp.getContent();

// Archive the date in three strings
month = str(month());
day = str(day());
year = str(year());
}

void draw(){
    // Begin while loop to get weather conditions
    while(j < 1){ // Selectively compare characters at different points
in the 'conditions' string to assign an index value to
the weather condition and offer a custom suggestion to display to the
user
        char index = conditions.charAt(1);
        if(index == 'F'){
            if(conditions.charAt(2) == 'a'){
                // The conditions are 'Fair'
                message = 10;
                suggestion = "It's nice out! Hope you have sunglasses.";
                j = 1;
            }
            if(conditions.charAt(2) == 'o'){
                // The conditions are 'Fog'
                message = 70;
                suggestion = "Take your time on your commute today.";
                j = 1;
            }
            if(conditions.charAt(2) == 'r'){
                if(conditions.charAt(15) == 'S'){
                    // The conditions are 'Freezing Rain/Snow'
                    message = 93;
                    suggestion = "Don't forget to de-ice your windshield and
drive carefully.";
                    j = 1;
                }
                else{
                    // The conditions are 'Freezing Rain'
                    message = 90;
                    suggestion = "You'll need your winter coat and umbrella for
this one.";
                    j = 1;
                }
            }
        }
        if(index == 'C'){
            // The conditions are 'Clear'
            message = 15;
            suggestion = "Enjoy the sun today!";
            j = 1;
        }
    }
}

```

```

if(index == 'O'){
    // The conditions are 'Overcast'
    message = 20;
    suggestion = "You're not gonna need sunglasses.";
    j = 1;
}
if(index == 'A'){
    // The conditions are 'A Few Clouds'
    message = 21;
    suggestion = "Taking a cap with you might be a good idea.";
    j = 1;
}
if(index == 'P'){
    // The conditions are 'Partly Cloudy'
    suggestion = "Sunglasses might be helpful today.";
    message = 22;
    j = 1;
}
if(index == 'M'){
    // The conditions are 'Mostly cloudy'
    suggestion = "I don't think you'll need a cap to go outside.";
    message = 23;
    j = 1;
}
if(index == 'R'){
    if(conditions.charAt(7) == 'h'){
        // The conditions are 'Rain Showers'
        message = 31;
        suggestion = "Take a rain slicker with you when you go out.";
        j = 1;
    }
    if(conditions.charAt(8) == 'o'){
        // The conditions are 'Rain/Snow'
        message = 94;
        suggestion = "You'll need a parka AND an umbrella.";
        j = 1;
    }
    if(conditions.charAt(6) == 'I'){
        // The conditions are 'Rain/Ice'
        message = 60;
        suggestion = "Be really careful driving out there!";
        j = 1;
    }
    else{
        // The conditions are 'Rain'
        message = 30;
        suggestion = "Don't forget your umbrella today.";
        j = 1;
    }
}
if(index == 'T'){
    // The conditions are 'Thunderstorm'

```

```

message = 40;
suggestion = "Stay inside and be safe.";
j = 1;
}
if(index == 'H'){
    // The conditions are 'Hail'
    message = 63;
    suggestion = "Wait until this passes to go out.";
    j = 1;
}
if(index == 'S'){
    if(conditions.charAt(6) == 'S'){
        // The conditions are 'Snow Showers'
        message = 53;
        suggestion = "The roads could be slick today!";
        j = 1;
    }
    else{
        // The conditions are 'Snow'
        message = 50;
        suggestion = "I'd stay indoors if possible.";
        j = 1;
    }
}
if(index == 'B'){
    if(conditions.charAt(9) == 'S'){
        // The conditions are 'Blowing Snow'
        message = 56;
        suggestion = "If you have to drive, be careful!";
        j = 1;
    }
    else{
        // The conditions are 'Breezy'
        message = 81;
        suggestion = "Do you have a hoodie you could wear today?";
        j = 1;
    }
}
if(index == 'W'){
    // The conditions are 'Windy'
    message = 80;
    suggestion = "A windbreaker would probably be a good idea.";
    j = 1;
}
if(index == 'I'){
    // The conditions are 'Ice Pellets'
    message = 64;
    suggestion = "Don't drive today if you don't have to!";
    j = 1;
}
if(index == 'L'){
    // The word is 'Light'

```

```

if(conditions.charAt(7) == 'R'){
    if(conditions.charAt(12) == 'S'){
        // The conditions are 'Light Rain/Snow'
        message = 95;
        suggestion = "Be cautious while driving.";
        j = 1;
    }
    if(conditions.charAt(12) == 'I'){
        // The conditions are 'Light Rain/Ice'
        message = 61;
        suggestion = "Don't go too fast on your commute, it's icy!";
        j = 1;
    }
    if(conditions.charAt(13) == 'h'){
        // The conditions are 'Light Rain Showers'
        message = 32;
        suggestion = "You could probably be okay without an
umbrella.";
        j = 1;
    }
    else{
        // The conditions are 'Light Rain'
        message = 34;
        suggestion = "Pack an umbrella just in case.";
        j = 1;
    }
}
if(conditions.charAt(7) == 'F'){
    // The conditions are 'Light Freezing Rain'
    message = 91;
    suggestion = "Be careful on the roads.";
    j = 1;
}
if(conditions.charAt(7) == 'S'){
    if(conditions.charAt(12) == 'S'){
        // The conditions are 'Light Snow Showers'
        message = 54;
        suggestion = "I think you'll be fine if you need to go out
today.";
        j = 1;
    }
    else{
        // The conditions are 'Light Snow'
        message = 51;
        suggestion = "The roads may be a little slick today, be
careful.";
        j = 1;
    }
}
if(conditions.charAt(7) == 'T'){
    // The conditions are 'Light Thunderstorm'
    message = 41;

```

```

        suggestion = "Take an umbrella if you need to go out.";
        j = 1;
    }
    if(conditions.charAt(7) == 'I'){
        // The conditions are 'Light Ice Pellets'
        message = 65;
        suggestion = "You might have to de-ice your car today.";
        j = 1;
    }
}
if(conditions.charAt(5) == 'y'){
    // The word is 'Heavy'
    if(conditions.charAt(7) == 'R'){
        if(conditions.charAt(12) == 'S'){
            // The conditions are 'Heavy Rain/Snow'
            message = 96;
            suggestion = "I'd call off of work today, it's dangerous out
there.";
            j = 1;
        }
        if(conditions.charAt(12) == 'I'){
            // The conditions are 'Heavy Rain/Ice'
            message = 62;
            suggestion = "You should stay off the roads today.";
            j = 1;
        }
        if(conditions.charAt(13) == 'h'){
            // The conditions are 'Heavy Rain Showers'
            message = 33;
            suggestion = "It may look okay now, but take an umbrella.";
            j = 1;
        }
    }
    else{
        // The conditions are 'Heavy Rain'
        message = 35;
        suggestion = "DEFINITELY have an umbrella handy.";
        j = 1;
    }
}
    if(conditions.charAt(7) == 'F'){
        // The conditions are 'Heavy Freezing Rain'
        message = 92;
        suggestion = "Driving is probably not the best idea right
now.";
        j = 1;
    }
    if(conditions.charAt(7) == 'S'){
        if(conditions.charAt(12) == 'S'){
            // The conditions are 'Heavy Snow Showers'
            message = 55;
            suggestion = "You might get caught in a rough squall if you
have to drive today.";

```

```

        j = 1;
    }
    else{
        // The conditions are 'Heavy Snow'
        message = 52;
        suggestion = "You'll probably be snowed in today.";
        j = 1;
    }
}
if(conditions.charAt(7) == 'T'){
    // The conditions are 'Heavy Thunderstorm'
    message = 42;
    suggestion = "Stay indoors and away from windows until this
blows over.";
    j = 1;
}
if(conditions.charAt(7) == 'I'){
    // The conditions are 'Heavy Ice Pellets'
    message = 66;
    suggestion = "Driving is probably a bad idea right now.";
    j = 1;
}
}
}
// Begin temperature loop
for(i = 1; i < 100; i++){ // Methodically compare each character in
'conditions' string to each character in the 'digit1' and 'digit2'
strings to determine the temperature
    int n;
    for(n = 0; n < 9; n++){
        char c = digit1.charAt(n);
        if(conditions.charAt(i) == c){
            temp1 = c;
        }
    }
    int m;
    for(m = 0; m < 10; m++){
        char t = digit2.charAt(m);
        if(conditions.charAt(i+1) == t){
            temp2 = t;
        }
    }
    if(temp1 != 0){
        break;
    }
}

// Convert temperature characters to a byte
char temps[] = {temp1, temp2};
String temp = new String(temps);
int temp3 = int(temp);
byte temp4 = byte(temp3);

```

```

    // Generate a different recommendation for the user depending on the
    temperature
    if(temp3 < 35){
        rec = "You might want to put on a coat, hat, and gloves.";
    }
    if(temp3 > 60){
        rec = "You can probably wear shorts today.";
    }
    if(temp3 > 35 && temp3 < 60){
        rec = "Consider a sweatshirt or light jacket before going out
today.";
    }

    // Send the message to the Redbot and wait for a response
    com.write(message);
    if(com.available() > 0){ // Create a custom message on the computer
screen after the Redbot has responded
        int arduino = com.read();

        // Build the background
        colorMode(RED, GREEN, BLUE);
        background(0, 106, 104);

        // Create the font
        font = createFont("Georgia", 32);
        textFont(font);

        // Generate the message
        textAlign(CENTER);
        textBuild(width*0.5);
    }
}

// Custom function to build computer screen message
void textBuild(float x){
    fill(255);
    text("Hello! Today's date is " + month + "/" + day + "/" + year, x,
height*0.2); // Current date
    fill(255);
    text(conditions, x, height*0.4); // Current weather
    fill(255);
    text(suggestion, x, height*0.6); // Weather recommendation
    fill(255);
    text(rec, x, height*0.8); // Temperature recommendation
}

```

D. Data Sheets (Redbot, Fio, MPU6050, and XBee respectively)

Features

- High Performance, Low Power AVR[®] 8-Bit Microcontroller
- Advanced RISC Architecture
 - 131 Powerful Instructions – Most Single Clock Cycle Execution
 - 32 x 8 General Purpose Working Registers
 - Fully Static Operation
 - Up to 20 MIPS Throughput at 20 MHz
 - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
 - 4/8/16/32K Bytes of In-System Self-Programmable Flash program memory
 - 256/512/1K Bytes EEPROM
 - 512/1K/1K/2K Bytes Internal SRAM
 - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
 - Data retention: 20 years at 85°C/100 years at 25°C⁽¹⁾
 - Optional Boot Code Section with Independent Lock Bits
 - In-System Programming by On-chip Boot Program
 - True Read-While-Write Operation
 - Programming Lock for Software Security
- Peripheral Features
 - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
 - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
 - Real Time Counter with Separate Oscillator
 - Six PWM Channels
 - 8-channel 10-bit ADC in TQFP and QFN/MLF package
 - Temperature Measurement
 - 6-channel 10-bit ADC in PDIP Package
 - Temperature Measurement
 - Programmable Serial USART
 - Master/Slave SPI Serial Interface
 - Byte-oriented 2-wire Serial Interface (Philips I²C compatible)
 - Programmable Watchdog Timer with Separate On-chip Oscillator
 - On-chip Analog Comparator
 - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
 - Power-on Reset and Programmable Brown-out Detection
 - Internal Calibrated Oscillator
 - External and Internal Interrupt Sources
 - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
 - 23 Programmable I/O Lines
 - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
 - 1.8 - 5.5V
- Temperature Range:
 - -40°C to 85°C
- Speed Grade:
 - 0 - 4 MHz@1.8 - 5.5V, 0 - 10 MHz@2.7 - 5.5V, 0 - 20 MHz @ 4.5 - 5.5V
- Power Consumption at 1 MHz, 1.8V, 25°C
 - Active Mode: 0.2 mA
 - Power-down Mode: 0.1 µA
 - Power-save Mode: 0.75 µA (Including 32 kHz RTC)



8-bit AVR[®]
Microcontroller
with 4/8/16/32K
Bytes In-System
Programmable
Flash

ATmega48A
ATmega48PA
ATmega88A
ATmega88PA
ATmega168A
ATmega168PA
ATmega328
ATmega328P

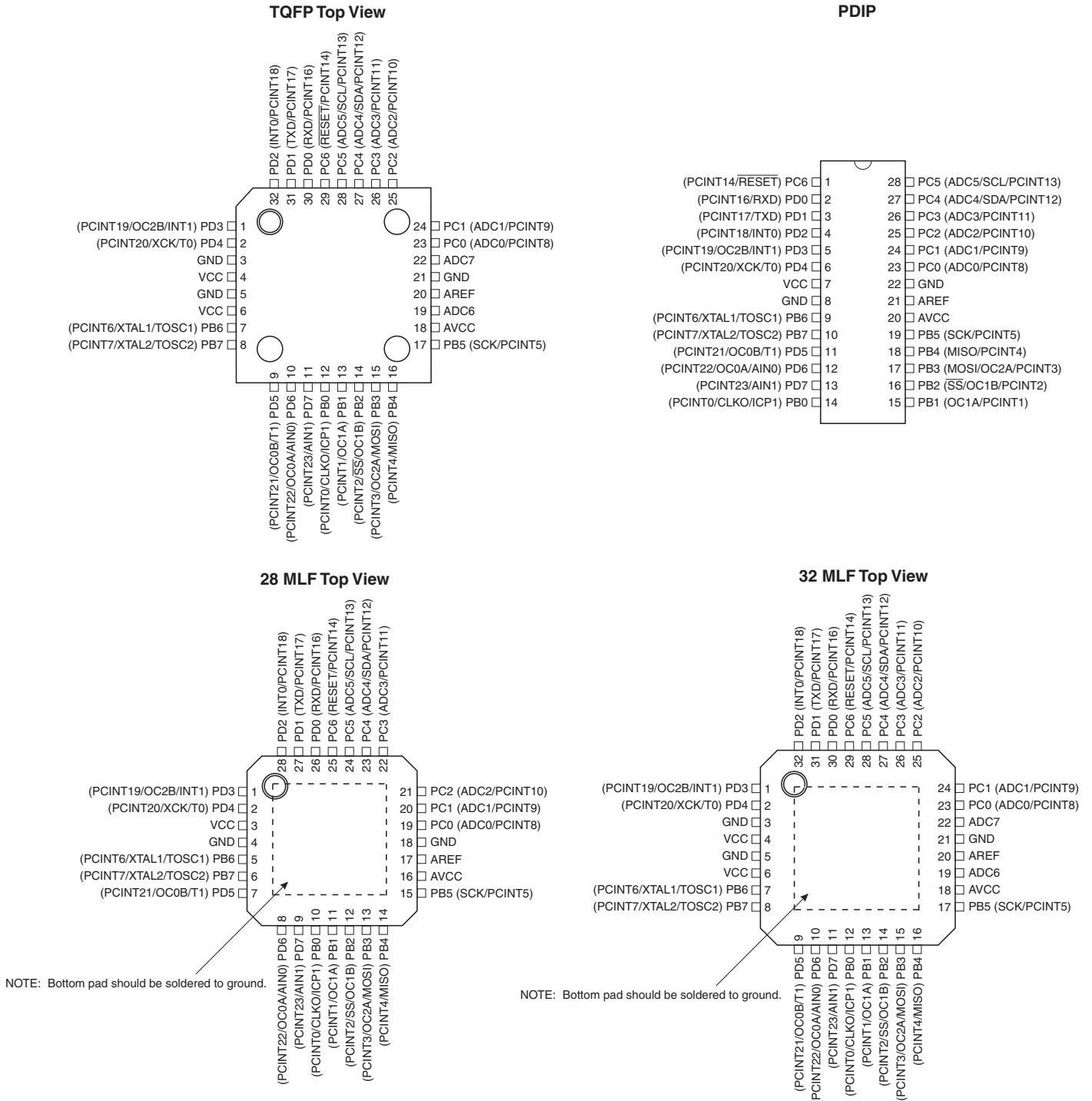
Summary

Rev. 8271BS-AVR-04/10



1. Pin Configurations

Figure 1-1. Pinout ATmega48A/48PA/88A/88PA/168A/168PA/328/328P



1.1 Pin Descriptions

1.1.1 VCC

Digital supply voltage.

1.1.2 GND

Ground.

1.1.3 Port B (PB7:0) XTAL1/XTAL2/TOSC1/TOSC2

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier.

If the Internal Calibrated RC Oscillator is used as chip clock source, PB7...6 is used as TOSC2...1 input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

The various special features of Port B are elaborated in [and "System Clock and Clock Options" on page 26](#).

1.1.4 Port C (PC5:0)

Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC5...0 output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

1.1.5 PC6/ $\overline{\text{RESET}}$

If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C.

If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. The minimum pulse length is given in [Table 28-12 on page 323](#). Shorter pulses are not guaranteed to generate a Reset.

The various special features of Port C are elaborated in ["Alternate Functions of Port C" on page 86](#).

1.1.6 Port D (PD7:0)

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

The various special features of Port D are elaborated in ["Alternate Functions of Port D" on page 89](#).

1.1.7 **AV_{CC}**

AV_{CC} is the supply voltage pin for the A/D Converter, PC3:0, and ADC7:6. It should be externally connected to V_{CC}, even if the ADC is not used. If the ADC is used, it should be connected to V_{CC} through a low-pass filter. Note that PC6...4 use digital supply voltage, V_{CC}.

1.1.8 **AREF**

AREF is the analog reference pin for the A/D Converter.

1.1.9 **ADC7:6 (TQFP and QFN/MLF Package Only)**

In the TQFP and QFN/MLF package, ADC7:6 serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

Miniature Single-Cell, Fully Integrated Li-Ion, Li-Polymer Charge Management Controllers

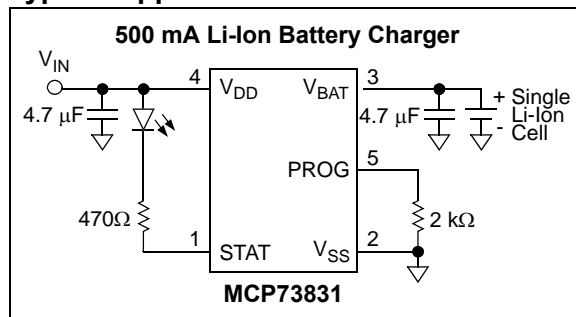
Features

- Linear Charge Management Controller:
 - Integrated Pass Transistor
 - Integrated Current Sense
 - Reverse Discharge Protection
- High Accuracy Preset Voltage Regulation: $\pm 0.75\%$
- Four Voltage Regulation Options:
 - 4.20V, 4.35V, 4.40V, 4.50V
- Programmable Charge Current: 15 mA to 500 mA
- Selectable Preconditioning:
 - 10%, 20%, 40%, or Disable
- Selectable End-of-Charge Control:
 - 5%, 7.5%, 10%, or 20%
- Charge Status Output
 - Tri-State Output - MCP73831
 - Open-Drain Output - MCP73832
- Automatic Power-Down
- Thermal Regulation
- Temperature Range: -40°C to $+85^{\circ}\text{C}$
- Packaging:
 - 8-Lead, 2 mm x 3 mm DFN
 - 5-Lead, SOT-23

Applications

- Lithium-Ion/Lithium-Polymer Battery Chargers
- Personal Data Assistants
- Cellular Telephones
- Digital Cameras
- MP3 Players
- Bluetooth Headsets
- USB Chargers

Typical Application



Description:

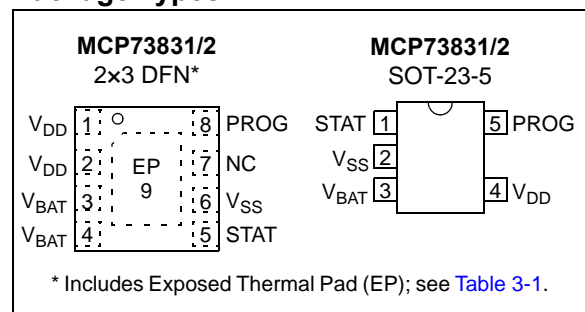
The MCP73831/2 devices are highly advanced linear charge management controllers for use in space-limited, cost-sensitive applications. The MCP73831/2 are available in an 8-Lead, 2 mm x 3 mm DFN package or a 5-Lead, SOT-23 package. Along with their small physical size, the low number of external components required make the MCP73831/2 ideally suited for portable applications. For applications charging from a USB port, the MCP73831/2 adhere to all the specifications governing the USB power bus.

The MCP73831/2 employ a constant-current/constant-voltage charge algorithm with selectable preconditioning and charge termination. The constant voltage regulation is fixed with four available options: 4.20V, 4.35V, 4.40V or 4.50V, to accommodate new, emerging battery charging requirements. The constant current value is set with one external resistor. The MCP73831/2 devices limit the charge current based on die temperature during high power or high ambient conditions. This thermal regulation optimizes the charge cycle time while maintaining device reliability.

Several options are available for the preconditioning threshold, preconditioning current value, charge termination value and automatic recharge threshold. The preconditioning value and charge termination value are set as a ratio, or percentage, of the programmed constant current value. Preconditioning can be disabled. Refer to **Section 1.0 "Electrical Characteristics"** for available options and the **"Product Identification System"** for standard options.

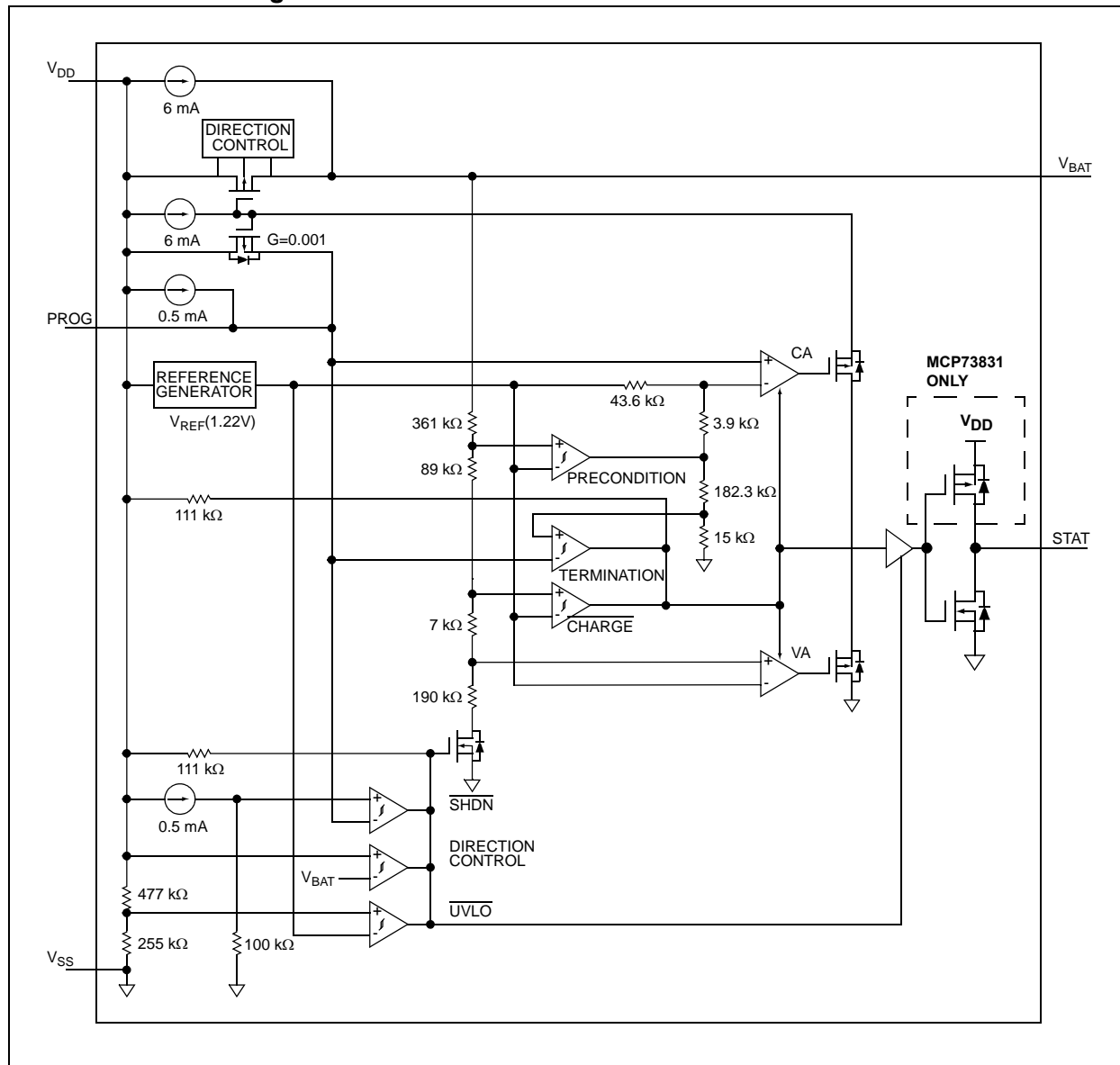
The MCP73831/2 devices are fully specified over the ambient temperature range of -40°C to $+85^{\circ}\text{C}$.

Package Types



MCP73831/2

Functional Block Diagram



1.0 ELECTRICAL CHARACTERISTICS

Absolute Maximum Ratings†

V_{DD} 7.0V
 All Inputs and Outputs w.r.t. V_{SS} -0.3 to ($V_{DD}+0.3$)V
 Maximum Junction Temperature, T_J Internally Limited
 Storage temperature -65°C to +150°C
 ESD protection on all pins:
 Human Body Model (1.5 k Ω in Series with 100 pF) ≥ 4 kV
 Machine Model (200 pF, No Series Resistance) 400V

† **Notice:** Stresses above those listed under “Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operational listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

DC CHARACTERISTICS

Electrical Specifications: Unless otherwise indicated, all limits apply for $V_{DD} = [V_{REG}(\text{typical}) + 0.3V]$ to 6V, $T_A = -40^\circ\text{C}$ to $+85^\circ\text{C}$. Typical values are at $+25^\circ\text{C}$, $V_{DD} = [V_{REG}(\text{typical}) + 1.0V]$

Parameters	Sym.	Min.	Typ.	Max.	Units	Conditions
Supply Input						
Supply Voltage	V _{DD}	3.75	—	6	V	
Supply Current	I _{SS}	—	510	1500	μA	Charging
		—	53	200	μA	Charge Complete, No Battery
		—	25	50	μA	PROG Floating
		—	1	5	μA	V _{DD} ≤ (V _{BAT} - 50 mV)
		—	0.1	2	μA	V _{DD} < V _{STOP}
UVLO Start Threshold	V _{START}	3.3	3.45	3.6	V	V _{DD} Low-to-High
UVLO Stop Threshold	V _{STOP}	3.2	3.38	3.5	V	V _{DD} High-to-Low
UVLO Hysteresis	V _{HYS}	—	70	—	mV	
Voltage Regulation (Constant-Voltage Mode)						
Regulated Output Voltage	V _{REG}	4.168	4.20	4.232	V	MCP7383X-2
		4.317	4.35	4.383	V	MCP7383X-3
		4.367	4.40	4.433	V	MCP7383X-4
		4.466	4.50	4.534	V	MCP7383X-5
						V _{DD} = [V _{REG} (typical)+1V] I _{OUT} = 10 mA T _A = -5°C to +55°C
Line Regulation	(Δ V _{BAT} /V _{BAT})/(Δ V _{DD})	—	0.09	0.30	%/V	V _{DD} = [V _{REG} (typical)+1V] to 6V, I _{OUT} = 10 mA
Load Regulation	Δ V _{BAT} /V _{BAT}	—	0.05	0.30	%	I _{OUT} = 10 mA to 50 mA V _{DD} = [V _{REG} (typical)+1V]
Supply Ripple Attenuation	PSRR	—	52	—	dB	I _{OUT} =10 mA, 10Hz to 1 kHz
		—	47	—	dB	I _{OUT} =10 mA, 10Hz to 10 kHz
		—	22	—	dB	I _{OUT} =10 mA, 10Hz to 1 MHz
Current Regulation (Fast Charge Constant-Current Mode)						
Fast Charge Current Regulation	I _{REG}	90	100	110	mA	PROG = 10 kΩ
		450	505	550	mA	PROG = 2.0 kΩ, Note 1
		12.5	14.5	16.5	mA	PROG = 67 kΩ
						T _A = -5°C to +55°C

Note 1: Not production tested. Ensured by design.

MCP73831/2

DC CHARACTERISTICS (CONTINUED)

Electrical Specifications: Unless otherwise indicated, all limits apply for $V_{DD} = [V_{REG}(\text{typical}) + 0.3V]$ to 6V, $T_A = -40^\circ\text{C}$ to $+85^\circ\text{C}$. Typical values are at $+25^\circ\text{C}$, $V_{DD} = [V_{REG}(\text{typical}) + 1.0V]$

Parameters	Sym.	Min.	Typ.	Max.	Units	Conditions
Preconditioning Current Regulation (Trickle Charge Constant-Current Mode)						
Precondition Current Ratio	$I_{\text{PREG}} / I_{\text{REG}}$	7.5	10	12.5	%	PROG = 2.0 k Ω to 10 k Ω
		15	20	25	%	PROG = 2.0 k Ω to 10 k Ω
		30	40	50	%	PROG = 2.0 k Ω to 10 k Ω
		—	100	—	%	No Preconditioning
						T _A = -5°C to +55°C
Precondition Voltage Threshold Ratio	$V_{\text{PTH}} / V_{\text{REG}}$	64	66.5	69	%	V _{BAT} Low-to-High
		69	71.5	74	%	V _{BAT} Low-to-High
Precondition Hysteresis	V _{PHYS}	—	110	—	mV	V _{BAT} High-to-Low
Charge Termination						
Charge Termination Current Ratio	$I_{\text{TERM}} / I_{\text{REG}}$	3.75	5	6.25	%	PROG = 2.0 k Ω to 10 k Ω
		5.6	7.5	9.4	%	PROG = 2.0 k Ω to 10 k Ω
		8.5	10	11.5	%	PROG = 2.0 k Ω to 10 k Ω
		15	20	25	%	PROG = 2.0 k Ω to 10 k Ω
						T _A = -5°C to +55°C
Automatic Recharge						
Recharge Voltage Threshold Ratio	$V_{\text{RTH}} / V_{\text{REG}}$	91.5	94.0	96.5	%	V _{BAT} High-to-Low
		94	96.5	99	%	V _{BAT} High-to-Low
Pass Transistor ON-Resistance						
ON-Resistance	R _{DS(ON)}	—	350	—	m Ω	V _{DD} = 3.75V, T _J = 105°C
Battery Discharge Current						
Output Reverse Leakage Current	$I_{\text{DISCHARGE}}$	—	0.15	2	μA	PROG Floating
		—	0.25	2	μA	V _{DD} Floating
		—	0.15	2	μA	V _{DD} < V _{STOP}
		—	-5.5	-15	μA	Charge Complete
Status Indicator – STAT						
Sink Current	I _{SINK}	—	—	25	mA	
Low Output Voltage	V _{OL}	—	0.4	1	V	I _{SINK} = 4 mA
Source Current	I _{SOURCE}	—	—	35	mA	
High Output Voltage	V _{OH}	—	V _{DD} -0.4	V _{DD} - 1	V	I _{SOURCE} = 4 mA (MCP73831)
Input Leakage Current	I _{LK}	—	0.03	1	μA	High-Impedance
PROG Input						
Charge Impedance Range	R _{PROG}	2	—	67	k Ω	
Minimum Shutdown Impedance	R _{PROG}	70	—	200	k Ω	
Automatic Power Down						
Automatic Power Down Entry Threshold	V _{PD(ENTER)}	V _{DD} <(V _{BAT} +20 mV)	V _{DD} <(V _{BAT} +50 mV)	—		3.5V ≤ V _{BAT} ≤ V _{REG} V _{DD} Falling
Automatic Power Down Exit Threshold	V _{PD(EXIT)}	—	V _{DD} <(V _{BAT} +150 mV)	V _{DD} <(V _{BAT} +200 mV)		3.5V ≤ V _{BAT} ≤ V _{REG} V _{DD} Rising
Thermal Shutdown						
Die Temperature	T _{SD}	—	150	—	°C	
Die Temperature Hysteresis	T _{SDHYS}	—	10	—	°C	

Note 1: Not production tested. Ensured by design.

AC CHARACTERISTICS

Electrical Specifications: Unless otherwise indicated, all limits apply for $V_{DD} = [V_{REG} \text{ (typical)} + 0.3V]$ to 12V, $T_A = -40^\circ\text{C}$ to $+85^\circ\text{C}$. Typical values are at $+25^\circ\text{C}$, $V_{DD} = [V_{REG} \text{ (typical)} + 1.0V]$

Parameters	Sym.	Min.	Typ.	Max.	Units	Conditions
UVLO Start Delay	t_{START}	—	—	5	ms	V_{DD} Low-to-High
Constant-Current Regulation						
Transition Time Out of Preconditioning	t_{DELAY}	—	—	1	ms	$V_{BAT} < V_{PTH}$ to $V_{BAT} > V_{PTH}$
Current Rise Time Out of Preconditioning	t_{RISE}	—	—	1	ms	I_{OUT} Rising to 90% of I_{REG}
Termination Comparator Filter	t_{TERM}	0.4	1.3	3.2	ms	Average I_{OUT} Falling
Charge Comparator Filter	t_{CHARGE}	0.4	1.3	3.2	ms	Average V_{BAT}
Status Indicator						
Status Output turn-off	t_{OFF}	—	—	200	μs	$I_{SINK} = 1 \text{ mA to } 0 \text{ mA}$
Status Output turn-on	t_{ON}	—	—	200	μs	$I_{SINK} = 0 \text{ mA to } 1 \text{ mA}$

TEMPERATURE SPECIFICATIONS

Electrical Specifications: Unless otherwise indicated, all limits apply for $V_{DD} = [V_{REG} \text{ (typical)} + 0.3V]$ to 12V. Typical values are at $+25^\circ\text{C}$, $V_{DD} = [V_{REG} \text{ (typical)} + 1.0V]$

Parameters	Sym.	Min.	Typ.	Max.	Units	Conditions
Temperature Ranges						
Specified Temperature Range	T_A	-40	—	+85	$^\circ\text{C}$	
Operating Temperature Range	T_J	-40	—	+125	$^\circ\text{C}$	
Storage Temperature Range	T_A	-65	—	+150	$^\circ\text{C}$	
Thermal Package Resistances						
5-Lead, SOT-23	θ_{JA}	—	230	—	$^\circ\text{C/W}$	4-Layer JC51-7 Standard Board, Natural Convection (Note 2)
8-Lead, 2 mm x 3 mm, DFN	θ_{JA}	—	76	—	$^\circ\text{C/W}$	4-Layer JC51-7 Standard Board, Natural Convection (Note 1)

Note 1: This represents the minimum copper condition on the PCB (Printed Circuit Board).

2: With large copper area on the PCB, the SOT-23-5 thermal resistance (θ_{JA}) can reach a typical value of 130°C/W or better.



MPU-6000/MPU-6050 Product Specification

Document Number: PS-MPU-6000A-00
Revision: 3.1
Release Date: 10/24/2011

1 Revision History

Revision Date	Revision	Description
11/24/2010	1.0	Initial Release
05/19/2011	2.0	For Rev C parts. Clarified wording in sections (3.2, 5.1, 5.2, 6.1-6.4, 6.6, 6.9, 7, 7.1-7.6, 7.11, 7.12, 7.14, 8, 8.2-8.4, 10.3, 10.4, 11, 12.2)
07/28/2011	2.1	Edited supply current numbers for different modes (section 6.4)
08/05/2011	2.2	Unit of measure for accelerometer sensitivity changed from LSB/mg to LSB/g
10/12/2011	2.3	Updated accelerometer self test specifications in Table 6.2. Updated package dimensions (section 11.2). Updated PCB design guidelines (section 11.3)
10/18/2011	3.0	For Rev D parts. Updated accelerometer specifications in Table 6.2. Updated accelerometer specification note (sections 8.2, 8.3, & 8.4). Updated qualification test plan (section 12.2).
10/24/2011	3.1	Edits for clarity Changed operating voltage range to 2.375V-3.46V Added accelerometer Intelligence Function increment value of 1mg/LSB (Section 6.2) Updated absolute maximum rating for acceleration (any axis, unpowered) from 0.3ms to 0.2ms (Section 6.9) Modified absolute maximum rating for Latch-up to Level A and $\pm 100\text{mA}$ (Section 6.9, 12.2)



2 Purpose and Scope

This product specification provides advanced information regarding the electrical specification and design related information for the MPU-6000™ and MPU-6050™ Motion Processing Unit™, collectively called the MPU-60X0™ or MPU™.

Electrical characteristics are based upon design analysis and simulation results only. Specifications are subject to change without notice. Final specifications will be updated based upon characterization of production silicon. For references to register map and descriptions of individual registers, please refer to the MPU-6000/MPU-6050 Register Map and Register Descriptions document.

ADVANCE INFORMATION



3 Product Overview

3.1 MPU-60X0 Overview

The MPU-60X0 Motion Processing Unit is the world's first motion processing solution with integrated 9-Axis sensor fusion using its field-proven and proprietary MotionFusion™ engine for handset and tablet applications, game controllers, motion pointer remote controls, and other consumer devices. The MPU-60X0 has an embedded 3-axis MEMS gyroscope, a 3-axis MEMS accelerometer, and a Digital Motion Processor™ (DMP™) hardware accelerator engine with an auxiliary I²C port that interfaces to 3rd party digital sensors such as magnetometers. When connected to a 3-axis magnetometer, the MPU-60X0 delivers a complete 9-axis MotionFusion output to its primary I²C or SPI port (SPI is available on MPU-6000 only). The MPU-60X0 combines acceleration and rotational motion plus heading information into a single data stream for the application. This MotionProcessing™ technology integration provides a smaller footprint and has inherent cost advantages compared to discrete gyroscope plus accelerometer solutions. The MPU-60X0 is also designed to interface with multiple non-inertial digital sensors, such as pressure sensors, on its auxiliary I²C port. The MPU-60X0 is a 2nd generation motion processor and is footprint compatible with the MPU-30X0 family.

The MPU-60X0 features three 16-bit analog-to-digital converters (ADCs) for digitizing the gyroscope outputs and three 16-bit ADCs for digitizing the accelerometer outputs. For precision tracking of both fast and slow motions, the parts feature a user-programmable gyroscope full-scale range of ± 250 , ± 500 , ± 1000 , and $\pm 2000^\circ/\text{sec}$ (dps) and a user-programmable accelerometer full-scale range of $\pm 2g$, $\pm 4g$, $\pm 8g$, and $\pm 16g$.

An on-chip 1024 Byte FIFO buffer helps lower system power consumption by allowing the system processor to read the sensor data in bursts and then enter a low-power mode as the MPU collects more data. With all the necessary on-chip processing and sensor components required to support many motion-based use cases, the MPU-60X0 uniquely supports a variety of advanced motion-based applications entirely on-chip. The MPU-60X0 thus enables low-power MotionProcessing in portable applications with reduced processing requirements for the system processor. By providing an integrated MotionFusion output, the DMP in the MPU-60X0 offloads the intensive MotionProcessing computation requirements from the system processor, minimizing the need for frequent polling of the motion sensor output.

Communication with all registers of the device is performed using either I²C at 400kHz or SPI at 1MHz (MPU-6000 only). For applications requiring faster communications, the sensor and interrupt registers may be read using SPI at 20MHz (MPU-6000 only). Additional features include an embedded temperature sensor and an on-chip oscillator with $\pm 1\%$ variation over the operating temperature range.

By leveraging its patented and volume-proven Nasiri-Fabrication platform, which integrates MEMS wafers with companion CMOS electronics through wafer-level bonding, InvenSense has driven the MPU-60X0 package size down to a revolutionary footprint of 4x4x0.9mm (QFN), while providing the highest performance, lowest noise, and the lowest cost semiconductor packaging required for handheld consumer electronic devices. The part features a robust 10,000g shock tolerance, and has programmable low-pass filters for the gyroscopes, accelerometers, and the on-chip temperature sensor.

For power supply flexibility, the MPU-60X0 operates from VDD power supply voltage range of 2.375V-3.46V. Additionally, the MPU-6050 provides a VLOGIC reference pin (in addition to its analog supply pin: VDD), which sets the logic levels of its I²C interface. The VLOGIC voltage may be $1.8V \pm 5\%$ or VDD.

The MPU-6000 and MPU-6050 are identical, except that the MPU-6050 supports the I²C serial interface only, and has a separate VLOGIC reference pin. The MPU-6000 supports both I²C and SPI interfaces and has a single supply pin, VDD, which is both the device's logic reference supply and the analog supply for the part. The table below outlines these differences:



MPU-6000/MPU-6050 Product Specification

Document Number: PS-MPU-6000A-00
Revision: 3.1
Release Date: 10/24/2011

Primary Differences between MPU-6000 and MPU-6050

Part / Item	MPU-6000	MPU-6050
VDD	2.375V-3.46V	2.375V-3.46V
VLOGIC	n/a	1.71V to VDD
Serial Interfaces Supported	I ² C, SPI	I ² C
Pin 8	/CS	VLOGIC
Pin 9	AD0/SDO	AD0
Pin 23	SCL/SCLK	SCL
Pin 24	SDA/SDI	SDA



4 Applications

- *BlurFree™* technology (for Video/Still Image Stabilization)
- *AirSign™* technology (for Security/Authentication)
- *TouchAnywhere™* technology (for “no touch” UI Application Control/Navigation)
- *MotionCommand™* technology (for Gesture Short-cuts)
- Motion-enabled game and application framework
- InstantGesture™ iG™ gesture recognition
- Location based services, points of interest, and dead reckoning
- Handset and portable gaming
- Motion-based game controllers
- 3D remote controls for Internet connected DTVs and set top boxes, 3D mice
- Wearable sensors for health, fitness and sports
- Toys

5 Features

5.1 Gyroscope Features

The triple-axis MEMS gyroscope in the MPU-60X0 includes a wide range of features:

- Digital-output X-, Y-, and Z-Axis angular rate sensors (gyroscopes) with a user-programmable full-scale range of ± 250 , ± 500 , ± 1000 , and $\pm 2000^\circ/\text{sec}$
- External sync signal connected to the FSYNC pin supports image, video and GPS synchronization
- Integrated 16-bit ADCs enable simultaneous sampling of gyros
- Enhanced bias and sensitivity temperature stability reduces the need for user calibration
- Improved low-frequency noise performance
- Digitally-programmable low-pass filter
- Gyroscope operating current: 3.6mA
- Standby current: 5 μ A
- Factory calibrated sensitivity scale factor

5.2 Accelerometer Features

The triple-axis MEMS accelerometer in MPU-60X0 includes a wide range of features:

- Digital-output triple-axis accelerometer with a programmable full scale range of $\pm 2g$, $\pm 4g$, $\pm 8g$ and $\pm 16g$
- Integrated 16-bit ADCs enable simultaneous sampling of accelerometers while requiring no external multiplexer
- Accelerometer normal operating current: 500 μ A
- Low power accelerometer mode current: 10 μ A at 1.25Hz, 20 μ A at 5Hz, 60 μ A at 20Hz, 110 μ A at 40Hz
- Orientation detection and signaling
- Tap detection
- User-programmable interrupts
- Free-fall interrupt
- High-G interrupt
- Zero Motion/Motion interrupt
- User self-test

5.3 Additional Features

The MPU-60X0 includes the following additional features:

- 9-Axis MotionFusion by the on-chip Digital Motion Processor (DMP)
- Auxiliary master I²C bus for reading data from external sensors (e.g., magnetometer)
- 3.9mA operating current when all 6 motion sensing axes and the DMP are enabled
- VDD supply voltage range of 2.375V-3.46V
- Flexible VLOGIC reference voltage supports multiple I²C interface voltages (MPU-6050 only)
- Smallest and thinnest QFN package for portable devices: 4x4x0.9mm
- Minimal cross-axis sensitivity between the accelerometer and gyroscope axes
- 1024 byte FIFO buffer reduces power consumption by allowing host processor to read the data in bursts and then go into a low-power mode as the MPU collects more data
- Digital-output temperature sensor
- User-programmable digital filters for gyroscope, accelerometer, and temp sensor
- 10,000 g shock tolerant
- 400kHz Fast Mode I²C for communicating with all registers
- 1MHz SPI serial interface for communicating with all registers (MPU-6000 only)



- 20MHz SPI serial interface for reading sensor and interrupt registers (MPU-6000 only)
- MEMS structure hermetically sealed and bonded at wafer level
- RoHS and Green compliant

5.4 MotionProcessing

- Internal Digital Motion Processing™ (DMP™) engine supports 3D MotionProcessing and gesture recognition algorithms
- The MPU-60X0 collects gyroscope and accelerometer data while synchronizing data sampling at a user defined rate. The total dataset obtained by the MPU-60X0 includes 3-Axis gyroscope data, 3-Axis accelerometer data, and temperature data. The MPU's calculated output to the system processor can also include heading data from a digital 3-axis third party magnetometer.
- The FIFO buffers the complete data set, reducing timing requirements on the system processor by allowing the processor burst read the FIFO data. After burst reading the FIFO data, the system processor can save power by entering a low-power sleep mode while the MPU collects more data.
- Programmable interrupt supports features such as gesture recognition, panning, zooming, scrolling, zero-motion detection, tap detection, and shake detection
- Digitally-programmable low-pass filters
- Low-power pedometer functionality allows the host processor to sleep while the DMP maintains the step count.

5.5 Clocking

- On-chip timing generator $\pm 1\%$ frequency variation over full temperature range
- Optional external clock inputs of 32.768kHz or 19.2MHz



6 Electrical Characteristics

6.1 Gyroscope Specifications

VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, T_A = 25°C

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
GYROSCOPE SENSITIVITY						
Full-Scale Range	FS_SEL=0		±250		°/s	
	FS_SEL=1		±500		°/s	
	FS_SEL=2		±1000		°/s	
	FS_SEL=3		±2000		°/s	
Gyroscope ADC Word Length			16		bits	
Sensitivity Scale Factor	FS_SEL=0		131		LSB/(°/s)	
	FS_SEL=1		65.5		LSB/(°/s)	
	FS_SEL=2		32.8		LSB/(°/s)	
	FS_SEL=3		16.4		LSB/(°/s)	
Sensitivity Scale Factor Tolerance	25°C	-3		+3	%	
Sensitivity Scale Factor Variation Over Temperature			±2		%	
Nonlinearity	Best fit straight line; 25°C		0.2		%	
Cross-Axis Sensitivity			±2		%	
GYROSCOPE ZERO-RATE OUTPUT (ZRO)						
Initial ZRO Tolerance	25°C		±20		°/s	
ZRO Variation Over Temperature	-40°C to +85°C		±20		°/s	
Power-Supply Sensitivity (1-10Hz)	Sine wave, 100mVpp; VDD=2.5V		0.2		°/s	
Power-Supply Sensitivity (10 - 250Hz)	Sine wave, 100mVpp; VDD=2.5V		0.2		°/s	
Power-Supply Sensitivity (250Hz - 100kHz)	Sine wave, 100mVpp; VDD=2.5V		4		°/s	
Linear Acceleration Sensitivity	Static		0.1		°/s/g	
GYROSCOPE NOISE PERFORMANCE	FS_SEL=0					
Total RMS Noise	DLPCCFG=2 (100Hz)		0.05		°/s-rms	
Low-frequency RMS noise	Bandwidth 1Hz to 10Hz		0.033		°/s-rms	
Rate Noise Spectral Density	At 10Hz		0.005		°/s/√Hz	
GYROSCOPE MECHANICAL FREQUENCIES						
X-Axis		30	33	36	kHz	
Y-Axis		27	30	33	kHz	
Z-Axis		24	27	30	kHz	
LOW PASS FILTER RESPONSE						
	Programmable Range	5		256	Hz	
OUTPUT DATA RATE						
	Programmable	4		8,000	Hz	
GYROSCOPE START-UP TIME						
ZRO Settling	DLPCCFG=0 to ±1°/s of Final		30		ms	



MPU-6000/MPU-6050 Product Specification

Document Number: PS-MPU-6000A-00
Revision: 3.1
Release Date: 10/24/2011

6.2 Accelerometer Specifications

VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, T_A = 25°C

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
ACCELEROMETER SENSITIVITY						
Full-Scale Range	AFS_SEL=0		±2		g	
	AFS_SEL=1		±4		g	
	AFS_SEL=2		±8		g	
	AFS_SEL=3		±16		g	
ADC Word Length	Output in two's complement format		16		bits	
Sensitivity Scale Factor	AFS_SEL=0		16,384		LSB/g	
	AFS_SEL=1		8,192		LSB/g	
	AFS_SEL=2		4,096		LSB/g	
	AFS_SEL=3		2,048		LSB/g	
Initial Calibration Tolerance			±3		%	
Sensitivity Change vs. Temperature	AFS_SEL=0, -40°C to +85°C		±0.02		%/°C	
Nonlinearity	Best Fit Straight Line		0.5		%	
Cross-Axis Sensitivity			±2		%	
ZERO-G OUTPUT						
Initial Calibration Tolerance ¹	X and Y axes		±50		mg	
	Z axis		±80		mg	
Zero-G Level Change vs. Temperature	X and Y axes, 0°C to +70°C		±35			
	Z axis, 0°C to +70°C		±60		mg	
SELF TEST RESPONSE			0.5		g	
NOISE PERFORMANCE						
Power Spectral Density	@10Hz, AFS_SEL=0 & ODR=1kHz		400		μg/√Hz	
LOW PASS FILTER RESPONSE						
	Programmable Range	5		260	Hz	
OUTPUT DATA RATE						
	Programmable Range	4		1,000	Hz	
INTELLIGENCE FUNCTION INCREMENT			1		mg/LSB	

1. Typical zero-g initial calibration tolerance value after MSL3 preconditioning

1. XBee®/XBee-PRO® RF Modules

The XBee and XBee-PRO RF Modules were engineered to meet IEEE 802.15.4 standards and support the unique needs of low-cost, low-power wireless sensor networks. The modules require minimal power and provide reliable delivery of data between devices.

The modules operate within the ISM 2.4 GHz frequency band and are pin-for-pin compatible with each other.



Key Features

Long Range Data Integrity

XBee

- Indoor/Urban: up to 100' (30 m)
- Outdoor line-of-sight: up to 300' (90 m)
- Transmit Power: 1 mW (0 dBm)
- Receiver Sensitivity: -92 dBm

XBee-PRO

- Indoor/Urban: up to 300' (90 m), 200' (60 m) for International variant
- Outdoor line-of-sight: up to 1 mile (1600 m), 2500' (750 m) for International variant
- Transmit Power: 63mW (18dBm), 10mW (10dBm) for International variant
- Receiver Sensitivity: -100 dBm

RF Data Rate: 250,000 bps

Advanced Networking & Security

Retries and Acknowledgements
DSSS (Direct Sequence Spread Spectrum)
Each direct sequence channels has over 65,000 unique network addresses available
Source/Destination Addressing
Unicast & Broadcast Communications
Point-to-point, point-to-multipoint and peer-to-peer topologies supported

Low Power

XBee

- TX Peak Current: 45 mA (@3.3 V)
- RX Current: 50 mA (@3.3 V)
- Power-down Current: < 10 μ A

XBee-PRO

- TX Peak Current: 250mA (150mA for international variant)
- TX Peak Current (RPSMA module only): 340mA (180mA for international variant)
- RX Current: 55 mA (@3.3 V)
- Power-down Current: < 10 μ A

ADC and I/O line support

Analog-to-digital conversion, Digital I/O
I/O Line Passing

Easy-to-Use

No configuration necessary for out-of box RF communications
Free X-CTU Software (Testing and configuration software)
AT and API Command Modes for configuring module parameters
Extensive command set
Small form factor

Worldwide Acceptance

FCC Approval (USA) Refer to Appendix A [p64] for FCC Requirements.
Systems that contain XBee®/XBee-PRO® RF Modules inherit Digi Certifications.

ISM (Industrial, Scientific & Medical) **2.4 GHz frequency band**

Manufactured under **ISO 9001:2000** registered standards

XBee®/XBee-PRO® RF Modules are optimized for use in the United States, Canada, Australia, Japan, and Europe. Contact Digi for complete list of government agency approvals.



Specifications

Table 1-01. Specifications of the XBee®/XBee-PRO® RF Modules

Specification	XBee	XBee-PRO
Performance		
Indoor/Urban Range	Up to 100 ft (30 m)	Up to 300 ft. (90 m), up to 200 ft (60 m) International variant
Outdoor RF line-of-sight Range	Up to 300 ft (90 m)	Up to 1 mile (1600 m), up to 2500 ft (750 m) international variant
Transmit Power Output (software selectable)	1mW (0 dBm)	63mW (18dBm)* 10mW (10 dBm) for International variant
RF Data Rate	250,000 bps	250,000 bps
Serial Interface Data Rate (software selectable)	1200 bps - 250 kbps (non-standard baud rates also supported)	1200 bps - 250 kbps (non-standard baud rates also supported)
Receiver Sensitivity	-92 dBm (1% packet error rate)	-100 dBm (1% packet error rate)
Power Requirements		
Supply Voltage	2.8 – 3.4 V	2.8 – 3.4 V
Transmit Current (typical)	45mA (@ 3.3 V)	250mA (@3.3 V) (150mA for international variant) RPSMA module only: 340mA (@3.3 V) (180mA for international variant)
Idle / Receive Current (typical)	50mA (@ 3.3 V)	55mA (@ 3.3 V)
Power-down Current	< 10 μ A	< 10 μ A
General		
Operating Frequency	ISM 2.4 GHz	ISM 2.4 GHz
Dimensions	0.960" x 1.087" (2.438cm x 2.761cm)	0.960" x 1.297" (2.438cm x 3.294cm)
Operating Temperature	-40 to 85° C (industrial)	-40 to 85° C (industrial)
Antenna Options	Integrated Whip, Chip or U.FL Connector, RPSMA Connector	Integrated Whip, Chip or U.FL Connector, RPSMA Connector
Networking & Security		
Supported Network Topologies	Point-to-point, Point-to-multipoint & Peer-to-peer	
Number of Channels (software selectable)	16 Direct Sequence Channels	12 Direct Sequence Channels
Addressing Options	PAN ID, Channel and Addresses	PAN ID, Channel and Addresses
Agency Approvals		
United States (FCC Part 15.247)	OUR-XBEE	OUR-XBEEPRO
Industry Canada (IC)	4214A XBEE	4214A XBEEPRO
Europe (CE)	ETSI	ETSI (Max. 10 dBm transmit power output)*
Japan	R201WW07215214	R201WW08215111 (Max. 10 dBm transmit power output)*
Australia	C-Tick	C-Tick

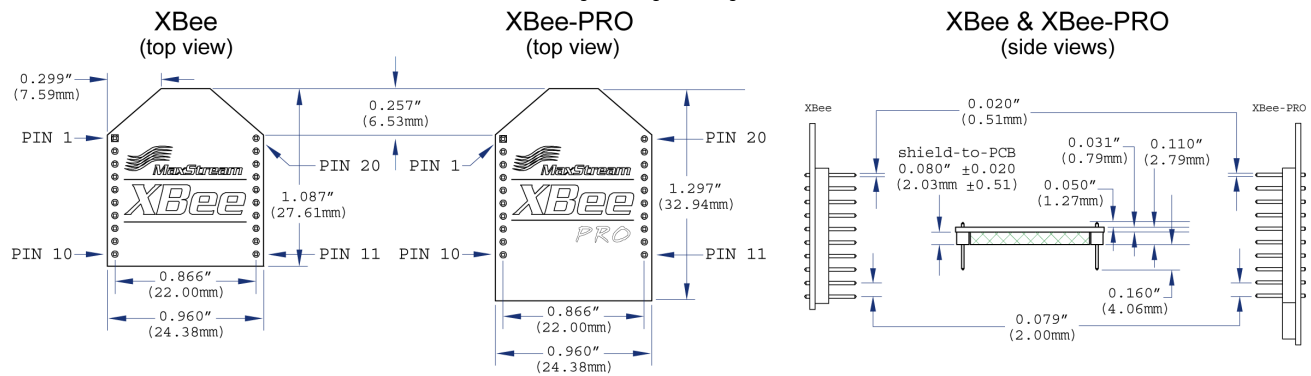
* See Appendix A for region-specific certification requirements.

Antenna Options: The ranges specified are typical when using the integrated Whip (1.5 dBi) and Dipole (2.1 dBi) antennas. The Chip antenna option provides advantages in its form factor; however, it typically yields shorter range than the Whip and Dipole antenna options when transmitting outdoors. For more information, refer to the "XBee Antennas" Knowledgebase Article located on Digi's Support Web site

Mechanical Drawings

Figure 1-01. Mechanical drawings of the XBee®/XBee-PRO® RF Modules (antenna options not shown)

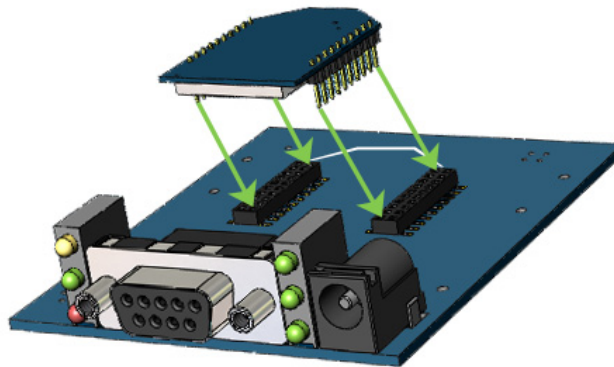
The XBee and XBee-PRO RF Modules are pin-for-pin compatible.



Mounting Considerations

The XBee®/XBee-PRO® RF Module was designed to mount into a receptacle (socket) and therefore does not require any soldering when mounting it to a board. The XBee Development Kits contain RS-232 and USB interface boards which use two 20-pin receptacles to receive modules.

Figure 1-02. XBee Module Mounting to an RS-232 Interface Board.



The receptacles used on Digi development boards are manufactured by Century Interconnect. Several other manufacturers provide comparable mounting solutions; however, Digi currently uses the following receptacles:

- Through-hole single-row receptacles -
Samtec P/N: MMS-110-01-L-SV (or equivalent)
- Surface-mount double-row receptacles -
Century Interconnect P/N: CPRMSL20-D-0-1 (or equivalent)
- Surface-mount single-row receptacles -
Samtec P/N: SMM-110-02-SM-S

Digi also recommends printing an outline of the module on the board to indicate the orientation the module should be mounted.

Pin Signals

Figure 1-03. XBee®/XBee-PRO® RF Module Pin Numbers

(top sides shown - shields on bottom)

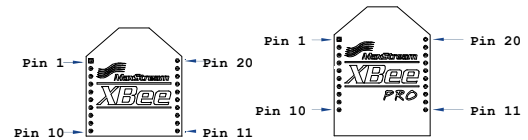


Table 1-02. Pin Assignments for the XBee and XBee-PRO Modules

(Low-asserted signals are distinguished with a horizontal line above signal name.)

Pin #	Name	Direction	Description
1	VCC	-	Power supply
2	DOUT	Output	UART Data Out
3	DIN / <u>CONFIG</u>	Input	UART Data In
4	DO8*	Output	Digital Output 8
5	<u>RESET</u>	Input	Module Reset (reset pulse must be at least 200 ns)
6	PWM0 / RSSI	Output	PWM Output 0 / RX Signal Strength Indicator
7	PWM1	Output	PWM Output 1
8	[reserved]	-	Do not connect
9	<u>DTR</u> / SLEEP_RQ / DI8	Input	Pin Sleep Control Line or Digital Input 8
10	GND	-	Ground
11	AD4 / DIO4	Either	Analog Input 4 or Digital I/O 4
12	<u>CTS</u> / DIO7	Either	Clear-to-Send Flow Control or Digital I/O 7
13	ON / <u>SLEEP</u>	Output	Module Status Indicator
14	VREF	Input	Voltage Reference for A/D Inputs
15	Associate / AD5 / DIO5	Either	Associated Indicator, Analog Input 5 or Digital I/O 5
16	<u>RTS</u> / AD6 / DIO6	Either	Request-to-Send Flow Control, Analog Input 6 or Digital I/O 6
17	AD3 / DIO3	Either	Analog Input 3 or Digital I/O 3
18	AD2 / DIO2	Either	Analog Input 2 or Digital I/O 2
19	AD1 / DIO1	Either	Analog Input 1 or Digital I/O 1
20	AD0 / DIO0	Either	Analog Input 0 or Digital I/O 0

* Function is not supported at the time of this release

Design Notes:

- Minimum connections: VCC, GND, DOUT & DIN
- Minimum connections for updating firmware: VCC, GND, DIN, DOUT, RTS & DTR
- Signal Direction is specified with respect to the module
- Module includes a 50k Ω pull-up resistor attached to RESET
- Several of the input pull-ups can be configured using the PR command
- Unused pins should be left disconnected

Electrical Characteristics

Table 1-03. DC Characteristics (VCC = 2.8 - 3.4 VDC)

Symbol	Characteristic	Condition	Min	Typical	Max	Unit
V _{IL}	Input Low Voltage	All Digital Inputs	-	-	0.35 * VCC	V
V _{IH}	Input High Voltage	All Digital Inputs	0.7 * VCC	-	-	V
V _{OL}	Output Low Voltage	I _{OL} = 2 mA, VCC ≥ 2.7 V	-	-	0.5	V
V _{OH}	Output High Voltage	I _{OH} = -2 mA, VCC ≥ 2.7 V	VCC - 0.5	-	-	V
I _{IN}	Input Leakage Current	V _{IN} = VCC or GND, all inputs, per pin	-	0.025	1	μA
I _{OZ}	High Impedance Leakage Current	V _{IN} = VCC or GND, all I/O High-Z, per pin	-	0.025	1	μA
TX	Transmit Current	VCC = 3.3 V	-	45 (XBee) 215, 140 (PRO, Int)	-	mA
RX	Receive Current	VCC = 3.3 V	-	50 (XBee) 55 (PRO)	-	mA
PWR-DWN	Power-down Current	SM parameter = 1	-	< 10	-	μA

Table 1-04. ADC Characteristics (Operating)

Symbol	Characteristic	Condition	Min	Typical	Max	Unit
V _{REFH}	VREF - Analog-to-Digital converter reference range		2.08	-	V _{DDAD} *	V
I _{REF}	VREF - Reference Supply Current	Enabled	-	200	-	μA
		Disabled or Sleep Mode	-	< 0.01	0.02	μA
V _{INDC}	Analog Input Voltage ¹		V _{SSAD} - 0.3	-	V _{DDAD} + 0.3	V

1. Maximum electrical operating range, not valid conversion range.

* V_{DDAD} is connected to VCC.

Table 1-05. ADC Timing/Performance Characteristics¹

Symbol	Characteristic	Condition	Min	Typical	Max	Unit
R _{AS}	Source Impedance at Input ²		-	-	10	kΩ
V _{AIN}	Analog Input Voltage ³		V _{REFL}		V _{REFH}	V
RES	Ideal Resolution (1 LSB) ⁴	2.08V ≤ V _{DDAD} ≤ 3.6V	2.031	-	3.516	mV
DNL	Differential Non-linearity ⁵		-	±0.5	±1.0	LSB
INL	Integral Non-linearity ⁶		-	±0.5	±1.0	LSB
E _{ZS}	Zero-scale Error ⁷		-	±0.4	±1.0	LSB
F _{FS}	Full-scale Error ⁸		-	±0.4	±1.0	LSB
E _{IL}	Input Leakage Error ⁹		-	±0.05	±5.0	LSB
E _{TU}	Total Unadjusted Error ¹⁰		-	±1.1	±2.5	LSB

1. All ACCURACY numbers are based on processor and system being in WAIT state (very little activity and no IO switching) and that adequate low-pass filtering is present on analog input pins (filter with 0.01 μF to 0.1 μF capacitor between analog input and VREFL). Failure to observe these guidelines may result in system or microcontroller noise causing accuracy errors which will vary based on board layout and the type and magnitude of the activity.

Data transmission and reception during data conversion may cause some degradation of these specifications, depending on the number and timing of packets. It is advisable to test the ADCs in your installation if best accuracy is required.

2. R_{AS} is the real portion of the impedance of the network driving the analog input pin. Values greater than this amount may not fully charge the input circuitry of the ATD resulting in accuracy error.

3. Analog input must be between V_{REFL} and V_{REFH} for valid conversion. Values greater than V_{REFH} will convert to \$3FF.

4. The resolution is the ideal step size or 1LSB = (V_{REFH} - V_{REFL})/1024

5. Differential non-linearity is the difference between the current code width and the ideal code width (1LSB). The current code width is the difference in the transition voltages to and from the current code.

6. Integral non-linearity is the difference between the transition voltage to the current code and the adjusted ideal transition voltage for the current code. The adjusted ideal transition voltage is (Current Code - 1/2) * (1 / ((V_{REFH} + E_{FS}) - (V_{REFL} + E_{ZS}))).

7. Zero-scale error is the difference between the transition to the first valid code and the ideal transition to that code. The Ideal transition voltage to a given code is (Code - 1/2) * (1 / (V_{REFH} - V_{REFL})).

8. Full-scale error is the difference between the transition to the last valid code and the ideal transition to that code. The ideal transition voltage to a given code is (Code - 1/2) * (1 / (V_{REFH} - V_{REFL})).

9. Input leakage error is error due to input leakage across the real portion of the impedance of the network driving the analog pin. Reducing the impedance of the network reduces this error.

10. Total unadjusted error is the difference between the transition voltage to the current code and the ideal straight-line transfer function. This measure of error includes inherent quantization error (1/2LSB) and circuit error (differential, integral, zero-scale, and full-scale) error. The specified value of E_{TU} assumes zero E_{IL} (no leakage or zero real source impedance).

2. RF Module Operation

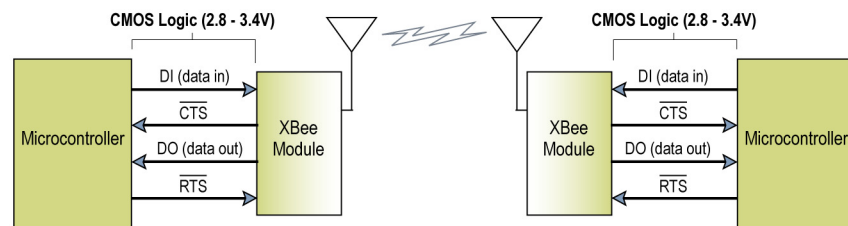
Serial Communications

The XBee®/XBee-PRO® RF Modules interface to a host device through a logic-level asynchronous serial port. Through its serial port, the module can communicate with any logic and voltage compatible UART; or through a level translator to any serial device (For example: Through a Digi proprietary RS-232 or USB interface board).

UART Data Flow

Devices that have a UART interface can connect directly to the pins of the RF module as shown in the figure below.

Figure 2-01. System Data Flow Diagram in a UART-interfaced environment
(Low-asserted signals distinguished with horizontal line over signal name.)

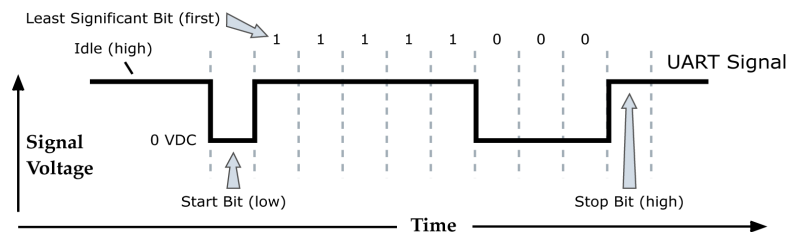


Serial Data

Data enters the module UART through the DI pin (pin 3) as an asynchronous serial signal. The signal should idle high when no data is being transmitted.

Each data byte consists of a start bit (low), 8 data bits (least significant bit first) and a stop bit (high). The following figure illustrates the serial bit pattern of data passing through the module.

Figure 2-02. UART data packet 0x1F (decimal number "31") as transmitted through the RF module
Example Data Format is 8-N-1 (bits - parity - # of stop bits)



Serial communications depend on the two UARTs (the microcontroller's and the RF module's) to be configured with compatible settings (baud rate, parity, start bits, stop bits, data bits).

The UART baud rate and parity settings on the XBee module can be configured with the BD and SB commands, respectively. See the command table in Chapter 3 for details.

Transparent Operation

By default, XBee®/XBee-PRO® RF Modules operate in Transparent Mode. When operating in this mode, the modules act as a serial line replacement - all UART data received through the DI pin is queued up for RF transmission. When RF data is received, the data is sent out the DO pin.

Serial-to-RF Packetization

Data is buffered in the DI buffer until one of the following causes the data to be packetized and transmitted:

1. No serial characters are received for the amount of time determined by the RO (Packetization Timeout) parameter. If RO = 0, packetization begins when a character is received.
2. The maximum number of characters that will fit in an RF packet (100) is received.
3. The Command Mode Sequence (GT + CC + GT) is received. Any character buffered in the DI buffer before the sequence is transmitted.

If the module cannot immediately transmit (for instance, if it is already receiving RF data), the serial data is stored in the DI Buffer. The data is packetized and sent at any RO timeout or when 100 bytes (maximum packet size) are received.

If the DI buffer becomes full, hardware or software flow control must be implemented in order to prevent overflow (loss of data between the host and module).

API Operation

API (Application Programming Interface) Operation is an alternative to the default Transparent Operation. The frame-based API extends the level to which a host application can interact with the networking capabilities of the module.

When in API mode, all data entering and leaving the module is contained in frames that define operations or events within the module.

Transmit Data Frames (received through the DI pin (pin 3)) include:

- RF Transmit Data Frame
- Command Frame (equivalent to AT commands)

Receive Data Frames (sent out the DO pin (pin 2)) include:

- RF-received data frame
- Command response
- Event notifications such as reset, associate, disassociate, etc.

The API provides alternative means of configuring modules and routing data at the host application layer. A host application can send data frames to the module that contain address and payload information instead of using command mode to modify addresses. The module will send data frames to the application containing status packets; as well as source, RSSI and payload information from received data packets.

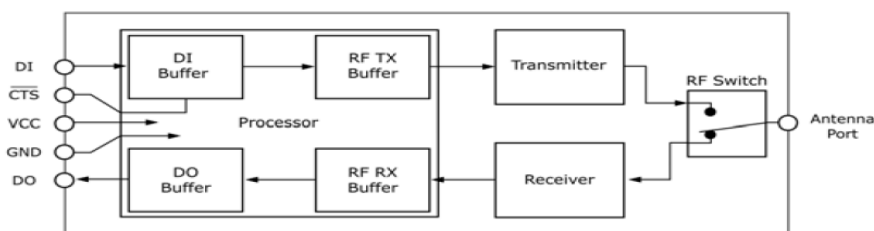
The API operation option facilitates many operations such as the examples cited below:

- > Transmitting data to multiple destinations without entering Command Mode
- > Receive success/failure status of each transmitted RF packet
- > Identify the source address of each received packet

To implement API operations, refer to API sections [p57].

Flow Control

Figure 2-03. Internal Data Flow Diagram



DI (Data In) Buffer

When serial data enters the RF module through the DI pin (pin 3), the data is stored in the DI Buffer until it can be processed.

Hardware Flow Control ($\overline{\text{CTS}}$). When the DI buffer is 17 bytes away from being full; by default, the module de-asserts $\overline{\text{CTS}}$ (high) to signal to the host device to stop sending data [refer to D7 (DIO7 Configuration) parameter]. $\overline{\text{CTS}}$ is re-asserted after the DI Buffer has 34 bytes of memory available.

How to eliminate the need for flow control:

1. Send messages that are smaller than the DI buffer size (202 bytes).
2. Interface at a lower baud rate [BD (Interface Data Rate) parameter] than the throughput data rate.

Case in which the DI Buffer may become full and possibly overflow:

If the module is receiving a continuous stream of RF data, any serial data that arrives on the DI pin is placed in the DI Buffer. The data in the DI buffer will be transmitted over-the-air when the module is no longer receiving RF data in the network.

Refer to the RO (Packetization Timeout), BD (Interface Data Rate) and D7 (DIO7 Configuration) command descriptions for more information.

DO (Data Out) Buffer

When RF data is received, the data enters the DO buffer and is sent out the serial port to a host device. Once the DO Buffer reaches capacity, any additional incoming RF data is lost.

Hardware Flow Control ($\overline{\text{RTS}}$). If $\overline{\text{RTS}}$ is enabled for flow control (D6 (DIO6 Configuration) Parameter = 1), data will not be sent out the DO Buffer as long as $\overline{\text{RTS}}$ (pin 16) is de-asserted.

Two cases in which the DO Buffer may become full and possibly overflow:

1. If the RF data rate is set higher than the interface data rate of the module, the module will receive data from the transmitting module faster than it can send the data to the host.
2. If the host does not allow the module to transmit data out from the DO buffer because of being held off by hardware or software flow control.

Refer to the D6 (DIO6 Configuration) command description for more information.

Glove-controlled RC Car with Autonomous Braking

December 19, 2014



Prepared by:
James Herring
Calvin Troxell

Prepared for:
Dr. Sommer
Michael Robinson

Table of Contents

1.0 Introduction.....	3
2.0 Mechanical Setup.....	3
2.1 DC Motor	5
2.2 ESC	5
2.3 Sonar Sensor	6
2.3 Hall Effect Sensor	7
2.4 Car Setup.....	8
2.5 Glove Setup.....	9
3.0 Software Setup	10
3.1 Pitch and Roll Measurement.....	10
3.2 XBee Communication.....	10
3.3 Motor Control	10
3.4 Steering Servo Control.....	11
3.5 Distance Measurement.....	11
3.6 RPM Measurement	11
3.7 Mode Filtering	11
3.8 Autobraking	11
Appendix A: Bill of Materials	14
Appendix B: Code.....	15
Fio Code.....	15
Uno Code	18
Appendix C: Datasheets.....	25

1.0 Introduction

Throughout the semester, we have been given the chance to explore the different capabilities of microcontrollers and other components. Being given the opportunity to pursue any project we desired was overwhelming at first, so we discussed different options that would require the implementation of the things we had learned in this course. Inspired by emerging technologies in the transportation industry, we decided to focus our ME 445 project on the modification of an RC car. First, we wanted to implement a unique way to control the RC car. Mimicking motion controls commonly found in today's video games, we aimed to control the car with our hands by way of an accelerometer. Additionally, we wanted the car to exhibit automatic braking functionality. This technology would require the use of multiple sensors in conjunction with coding that would prevent the car from crashing in walls or obstacles. The combination of these technologies challenged us to further explore the information covered in class while allowing us to have fun along the way.

2.0 Mechanical Setup

After proposing our project, we were notified that there were a few different RC cars available that were preexisting in the lab. We ultimately landed on the Exceed RC Electric SunFire Off Road Buggy. This car came equipped with a Brushed DC motor, an electric speed controller, and servos for steering capability. The car is also equipped with shocks and tire treads to protect it during our testing. This was important because we fully expected our car to experience many crashes during the course of our project.

We came into possession of the car with the main body removed, exposing the innards and components aforementioned. Aesthetics were an afterthought for us, and this made it possible to modify the car how we needed it. We also needed to house components that were not original to the RC car. Primarily, we needed to devise a way to mount our Arduino onto the car. We also needed to mount small breadboards to make wiring to our different components possible. Figure 1 shows how we installed the Arduino and other sensor onto the RC car.

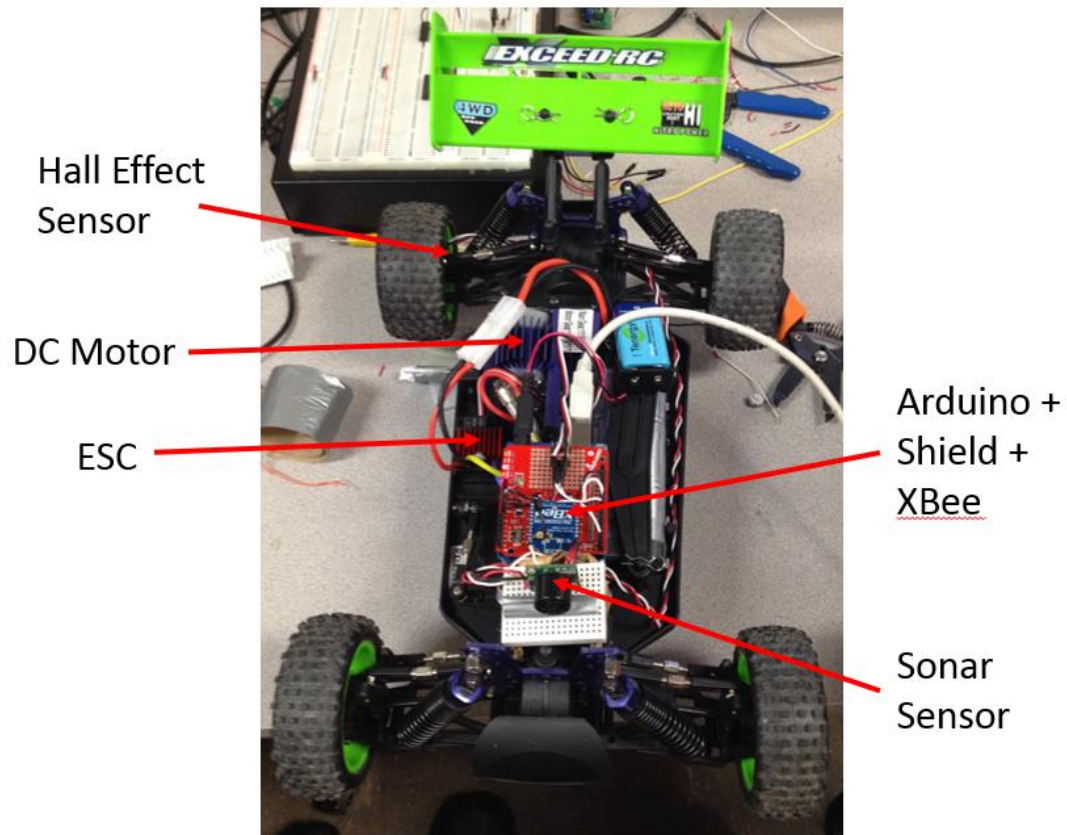


Figure 1: RC Car Component Overview

The glove that will control the RC car is powered by the Arduino Fio. The Fio is hooked up to an accelerometer on a mini-breadboard. This entire setup is attached to glove as shown in Figure 2.

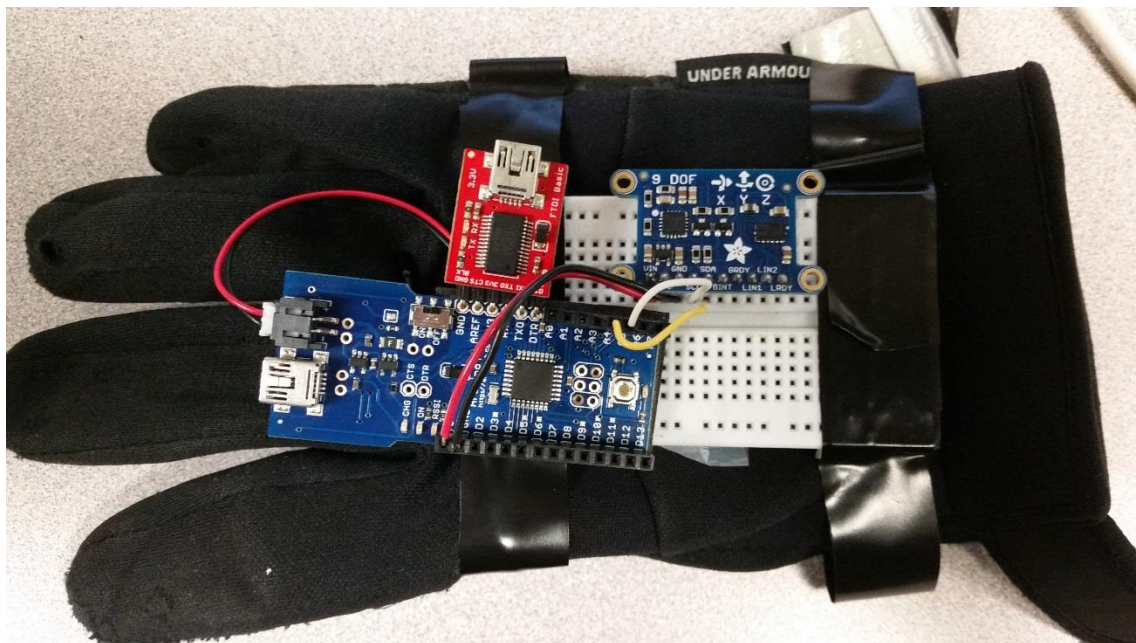


Figure 2: Glove Component Overview

2.1 DC Motor

The DC motor for this car is listed as a 540 brushed motor. Unfortunately, this was all the information we were able to pull from this motor. Data sheets exist for other 540 motors, but the number of turns for these 540 motors was different than the number of turns for our 540 motor. The motor was powered by a 7.2V 6-cell Ni-MH battery. This part also came with the car and was able to be fitted into a pre-determined position. When we received the car, the motor was plugged into an ESC, or electronic speed controller. We were informed that this device may be broken, and after some preliminary testing, we were able to confirm that the ESC was broken. Thankfully, we were able to obtain a new ESC that had better datasheets backing up its capabilities.

2.2 ESC

The ESC has 3 different running modes: Forward/Brake/Reverse, Forward/Brake, and Forward/Reverse. These modes can be chosen by moving the included ESC pins to the appropriate position.

Initially, we thought we were going to use the Forward/Brake/Reverse mode for our project. It made intuitive sense to us that we would need braking capability on our car if we were to have auto-braking functionality. The Forward/Brake/Reverse mode actually serves more as a safety measure for RC car hobbyists so that unintentional braking does not occur during steering maneuvers that may require some braking input. This mode enables a “double-click” method that requires two brake inputs for effective braking to take place. Braking in Forward/Brake/Reverse mode causes the car’s motor to essentially act as a generator and this causes the car to coast when the brakes are applied.

After consulting the information available pertaining to this ESC, we discovered that we could still effectively brake the car in Forward/Reverse mode. This is why you see the pin completely removed in Figure 3. The Forward/Reverse mode does not require the “double-click” method. Therefore, we were able to switch from forward to reverse much quicker. This was a complication that came about only because of how each running mode was named. The ESC still allows for braking in Forward/Reverse mode even though the name does not suggest so.



Figure 3: Close-up of ESC's Running Modes

2.3 Sonar Sensor

Another critical hardware component was the sonar sensor. We knew from the beginning that we would require some kind of distance sensing equipment. In the lab, we found a LV-MaxSonar-EZ sensor that was available for use. We quickly learned that there exists multiple different kinds of these sensors that vary in range and sensing capability. Our model, MB1010, was designated by a brown marking and is shown on our RC car in Figure 4. Testing the sensor showed us that the sensor would not be able to detect distances less than six inches. We figured this would be okay because our auto-braking system would have to apply the brakes at a distance much greater than six inches.

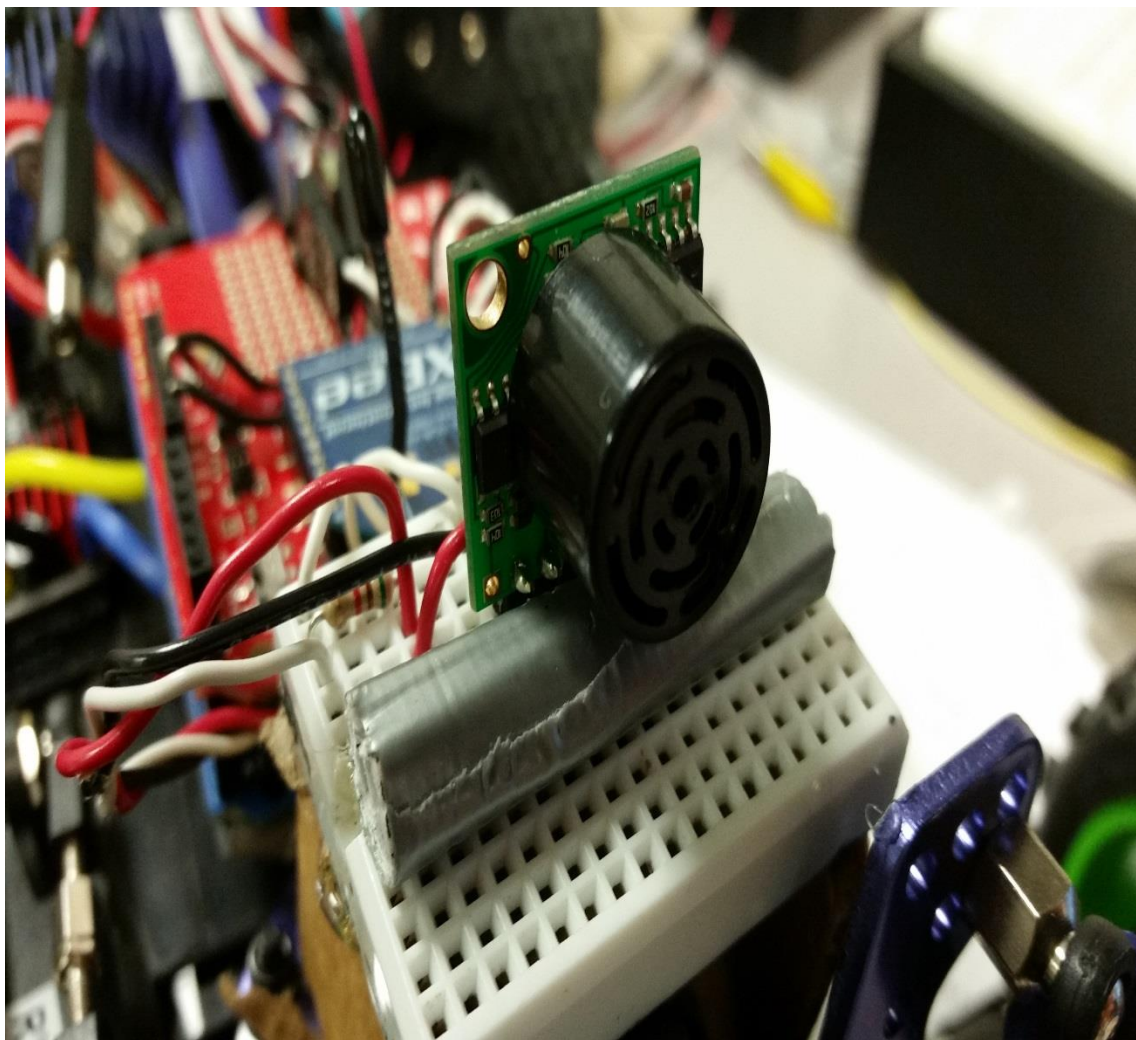


Figure 4: Sonar Sensor

We originally planned to take the derivative of the distance measurements to obtain our car's velocity. Even though this worked, the results were not resolute enough for our standards. Due to the nature of the project, we required much greater resolution for our velocity readings. For instance, taking velocity with the sonar sensor resulted in a minimum reading of 30 inches/second. After consulting with Mike Robinson, we decided that an alternate approach to velocity measurement was required.

2.3 Hall Effect Sensor

A Hall Effect sensor is attached to the axle on the inside of the rear right wheel. This sensor is able to detect changes in magnetic field around it, so a magnet was hot-glued onto the rotating wheel. Each time the wheel makes a whole revolution, the Hall Effect sensor picks up a signal. We were then able to multiply the frequency of these signals by the diameter and radius of the wheel to obtain velocity in ft/s. The hardest part of this process was placement of the sensor itself. The inside of the wheel is a very dynamic region where vibrations abound. After a few

unsuccessful attempts, we were able to fasten the sensor in a static location that was still able to pick up the magnet's field as shown in Figure 5.



Figure 5: Hall Effect Sensor

2.4 Car Setup

Of course, the heart of all of these components is the Arduino that provides power and receives data for the different instruments. We ended up requiring two Arduinos for this project. An Arduino Uno was used on the body of the car. On it, we attached a SparkFun XBee Shield so that an Xbee module could also be added. See Figure 6 for a visual representation of how each component was connected to the Arduino and shield.

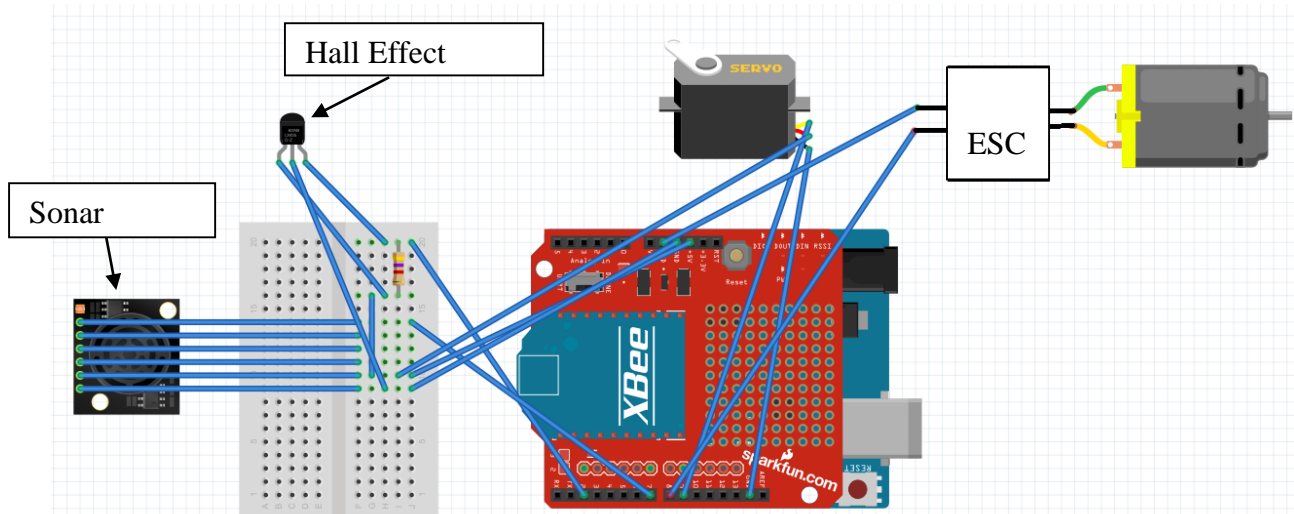


Figure 6: Overview of Car Setup

2.5 Glove Setup

We also used an Arduino Fio for control of the hand gestures that is seen in Figure 7. An accelerometer was used for the detection of hand movements. We used an Adafruit 9-DOF IMU Breakout that also came equipped with a gyroscope and compass sensor. This product is very popular, so it was easy to quickly learn how to use it. One thing we learned with this portion of the project was how to upload code to the Arduino Fio. A basic FTDI breakout was required to be connected to the Fio so that we could implement the code we had written.

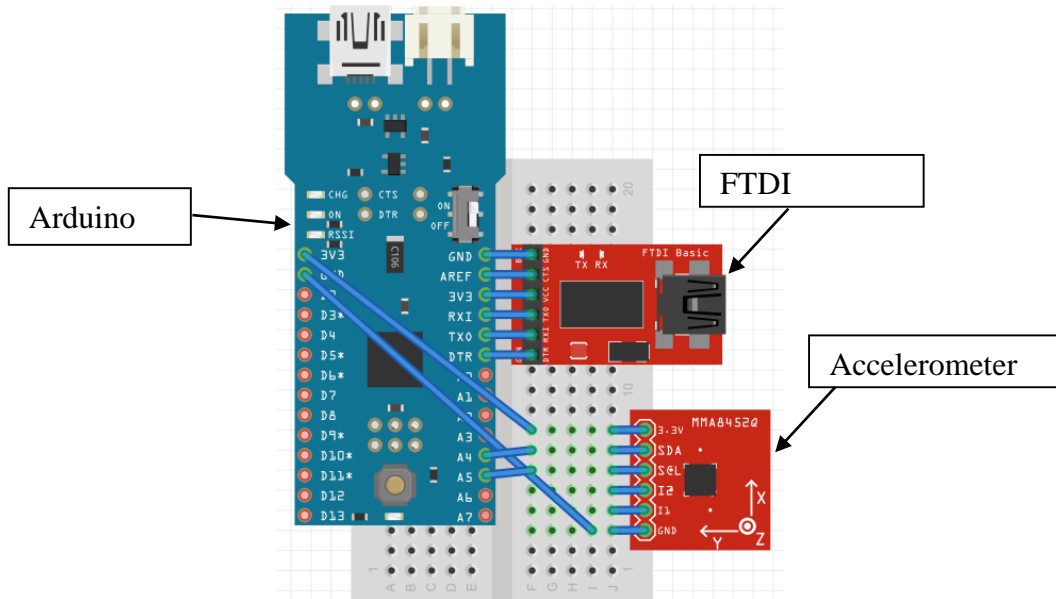


Figure 7: Overview of Hand Controller

3.0 Software Setup

The RC car is controlled by an Arduino Fio and an Arduino Uno. The Fio is used to measure the pitch and roll from an Adafruit 9-DOF IMU Breakout on the driver's hand. The pitch and roll values are then wirelessly sent by XBee to the Uno. The Uno uses the pitch and roll values to control the motor and steering servo. The autobraking functionality is controlled by the Uno. The Uno uses distance measurements from a MaxSonar Ultrasonic Rangefinder and rpm measurement from a Hall Effect sensor to calculate the minimum braking distance to avoid collision. When the RC car enters the minimum braking distance, the brakes are applied by the Uno.

3.1 Pitch and Roll Measurement

The pitch and roll measurements that are used to control the RC car are measured from an Adafruit 9-DOF IMU Breakout. These values are obtained utilizing slightly modified code from Adafruit. Prior to being sent by Xbee the values are filtered by mode. Mode filtering is explained in Section 3.7 Mode Filtering.

3.2 XBee Communication

Xbee communication utilizes the serial print function built into the Arduino programming language to send information wirelessly. To differentiate the pitch and roll values, the letter p is printed before pitch values, and the letter r is printed before roll values. Using an Xbee Arduino shield, the Uno receives pitch and roll values using the communicate function. The communicate function reads the serial connection and then identifies the value as either a pitch or roll.

3.3 Motor Control

The ESC is controlled using the servo library that is built into the Arduino programming language. The Uno sends the ESC a pulse-width modulation (pwm) with the writeMicroseconds command. Table 1 shows the range of pwm signals that the ESC accepts and the motor response for each signal. The values sent to the ESC come from the pitch values from the Fio. The Uno remaps these values to the 1000-2000 range that the ESC accepts. These values are then sent to the ESC using the drive function, which slightly increases the range of values that will activate the brake. This was done so that the pitch value did not have to be exactly zero to activate the RC car's brake.

Table 1: Motor Response to ESC Signals

ESC Signal	Motor Response
1000-1499	Reverse
1500	Brake
1501-2000	Forward

3.4 Steering Servo Control

The steering servo is also controlled using the servo library that is built into the Arduino programming language. The Uno sends the steering servo a value between 0 and 180 which correspondingly sets the angle of the servo shaft. The Uno maps roll values from the Fio to 0 to 180 prior to sending these values to the steering servo.

3.5 Distance Measurement

Values from the sonar sensor are measured by using the pulseIn command, which measures the pulse values from the sonar sensor. The pulses are converted from pulses to inches by dividing by a scale factor of 147 uS per inch as provided in the LV-MaxSonar-EZ Datasheet. The accuracy of these values was improved by using a mode filter.

3.6 RPM Measurement

The RPMs of the RC car's wheels is measured with a Hall Effect sensor. The Hall Effect sensor uses an interrupt function to measure the number of half revolutions made by a magnet mounted to the RC car's wheel. The RPMs are calculated by using the number of half revolutions divided by the difference between the time of the current RPM reading and the time of the last RPM reading. The speed is then calculated by multiplying the RPMs by the circumference of the RC car's wheel.

3.7 Mode Filtering

To clean up some of the noise on the pitch, roll, RPM, and sonar measurements, a mode filter was used. Each measured value is stored into an array. The mode function is then used on each array to determine the mode for each measurement.

3.8 Autobraking

The auto-braking system is designed to apply the brakes and stop the RC car before it hits an object by using measurements from the sonar sensor and RPM sensor. The minimum braking distance can be found by inserting the current velocity into the minimum braking distance equation. The Uno will constantly compare the sonar distance value against the minimum braking distance. When the minimum braking distance is less than the sonar distance value, the brakes will automatically be applied until the RPMs of the RC car are zero. When the RC car is closer than 12 inches to an object the maximum speed of the car will be restricted and the car will not automatically brake to allow for precision driving. When the RC car is driving in reverse auto-braking will be disabled. Figure 8 shows a comprehensive flow chart of the RC car's auto-braking system.

The minimum braking distance was calculated based on conservation of momentum. The following equation was developed that loosely models the RC car.

Equation 1: $mv^2 - mb_c d = 0$

The mv^2 term is the momentum of the RC car and the $mb_c d$ term is braking force. We simplified the braking force to equal the mass of the RC car multiplied by the braking constant and the distance for the RC car to come to a complete stop. We conducted tests to attempt to derive the braking constant; however, the value we derived ended up being much larger than the braking constant that was used in the final version of the program.

To find the minimum braking distance, we rearranged Equation 1 into the following form.

Equation 2: $d = \frac{v^2}{b_c}$

Equation 2 was modified to taking into account the inaccuracy of the sonar sensor by adding a six, which is the minimum resolution of the sonar sensor, into the equation. The final equation for minimum braking distance is shown below in Equation 3.

Equation 3: $d = \frac{v^2}{b_c} + 6$

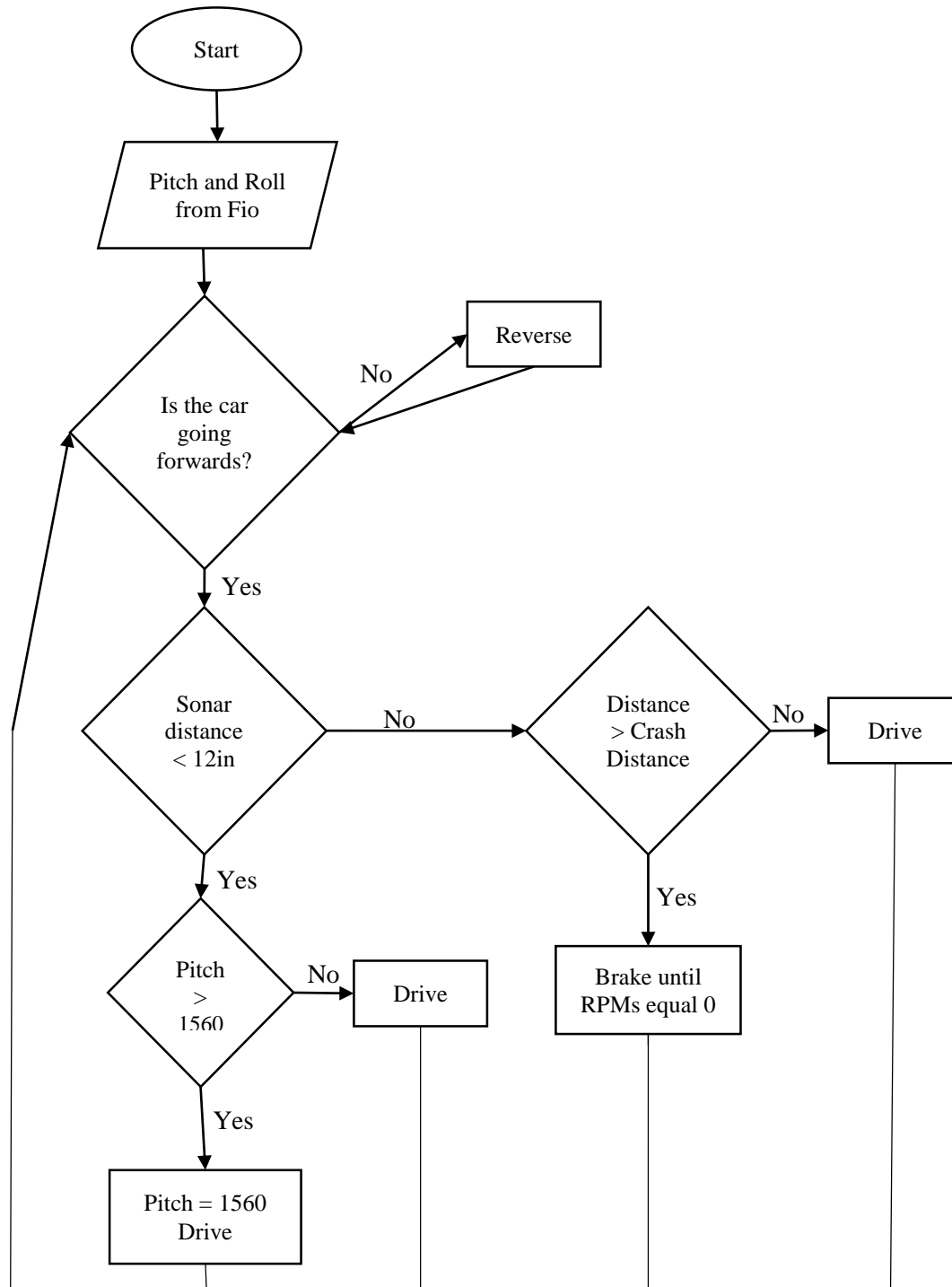


Figure 8: Code Flow Chart

Appendix A: Bill of Materials

Item	Manufacturer	Quantity	Price Per Unit	Total
1/10 2.4 GHz Exceed RC Electric SunFire RTR Off Road Buggy	Exceed RC	1	\$129.50	\$129.50
QuicRun 1060 Brushed Electronic Speed Controller	QuicRun	1	\$23.20	\$23.20
Arduino Uno	Arduino	1	\$28.90	\$28.90
Arduino Fio	Arduino	1	\$24.95	\$24.95
Xbee Shield	SparkFun	1	\$14.95	\$14.95
Xbee RF Modules with Wire Antenna	Digi International Inc.	2	\$ 27.95	\$55.90
Xbee Explorer Dongle	SparkFun	1	\$29.95	\$29.95
FTDI Basic Breakout - 5V	NKC Electronics	1	\$15.95	\$15.95
9-DOF IMU Breakout Accelerometer	Adafruit	1	\$19.95	\$19.95
MB1010 LV-MaxSonar-EZ1	MaxBotix	1	\$29.92	\$29.92
U18 Hall effect sensor w/ magnet	CuteDigi	1	\$1.99	\$1.99
			Total	\$375.16

Appendix B: Code

This section contains the code used on both the Fio and Uno to control the RC car.

Fio Code

```
// This code is the final version of code on the Fio. It is used to drive the RC car.
// This code works in conjunction with CarUno_Final on the Uno.
// This code is based on code from Adafruit for the Adafruit 9-DOF IMU Breakout.
// Written by Jimmy Herring & Calvin Troxell
// Last edited: CT 12/16/14 3:25pm

// Include libraries; these libraries are all necessary for the Adafruit 9-DOF IMU Breakout to run
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM303_U.h>
#include <Adafruit_9DOF.h>

/* Assign a unique ID to the sensors */
Adafruit_9DOF      dof  = Adafruit_9DOF();
Adafruit_LSM303_Accel_Unified accel = Adafruit_LSM303_Accel_Unified(30301);
Adafruit_LSM303_Mag_Unified  mag  = Adafruit_LSM303_Mag_Unified(30302);

/* Update this with the correct SLP for accurate altitude measurements */
float seaLevelPressure = SENSORS_PRESSURE_SEALEVELHPA;

// The following variables are for pitch mode filtering
int pitch_array_size = 3;    // Quantity of values used in mode calculation, must be an odd
                             // number
int pitch_value[] = {0,0,0}; // An empty array to hold pitch readings
int pitch_mode;              // Mode of pitch readings

// The following variables are for roll mode filtering
int roll_value[] = {0,0,0}; // An empty array to hold roll readings
int roll_mode;              // Mode of roll readings

/*****
/*!
  @brief Initialises all the sensors used by this example
*/
*****/
void initSensors()
{
  if(!accel.begin())
  {
    /* There was a problem detecting the LSM303 ... check your connections */
  }
}
```

```

    Serial.println(F("Oops, no LSM303 detected ... Check your wiring!"));
    while(1);
}
if(!mag.begin())
{
    /* There was a problem detecting the LSM303 ... check your connections */
    Serial.println("Oops, no LSM303 detected ... Check your wiring!");
    while(1);
}
}

/*****
*/

*/
/*****
void setup(void)
{
    Serial.begin(9600);
    //Serial.println(F("Adafruit 9 DOF Pitch/Roll/Heading Example")); Serial.println("");
    /* Initialise the sensors */
    initSensors();
    Serial.write("!");
}

/*****
*/
@brief Constantly check the roll/pitch/heading/altitude/temperature
*/
/*****
void loop(void)
{
    sensors_event_t accel_event;
    sensors_event_t mag_event;
    sensors_vec_t  orientation;

    /* Calculate pitch and roll from the raw accelerometer data */
    accel.getEvent(&accel_event);
    if (dof.accelGetOrientation(&accel_event, &orientation))
    {
        /* 'orientation' should have valid .roll and .pitch fields */

        // Pitch and roll values from the Adafruit 9-DOF IMU Breakout
        // are stored in an array for mode filtering.
        for(int i = 0; i < pitch_array_size; i++)
        {

```

```

    pitch_value[i] = orientation.pitch;
    roll_value[i] = orientation.roll;
}
roll_mode = mode(roll_value,pitch_array_size); // Mode of the roll array is found
pitch_mode = mode(pitch_value,pitch_array_size); // Mode of the pitch array is found

// The code below serially prints over XBee.
// An identifying letter is printed before the data so that the Uno can differeniates the data
// with the communicate function.
Serial.print("r"); // The identifying letter for roll
Serial.print(roll_mode); // Roll value
Serial.print("p"); // The identifying letter for pitch
Serial.println(pitch_mode); // Pitch value
delay(200); // This delay is included so that the Uno is checking for data more
quickly
// than the Fio is sending it. If a delay of less than 200 is included servo
// performace is jittery and spordadic.
}
}

//Mode function, returning the mode or median.
int mode(int *x,int n){

    int i = 0;
    int count = 0;
    int maxCount = 0;
    int mode = 0;
    int bimodal;
    int prevCount = 0;
    while(i<(n-1)){
        prevCount=count;
        count=0;
        while(x[i]==x[i+1]){
            count++;
            i++;
        }
        if(count>prevCount&count>maxCount){
            mode=x[i];
            maxCount=count;
            bimodal=0;
        }
        if(count==0){
            i++;
        }
        if(count==maxCount){//If the dataset has 2 or more modes.
            bimodal=1;

```

```

    }
    if(mode==0||bimodal==1){//Return the median if there is no mode.
        mode=x[(n/2)];
    }
    return mode;
}
}

```

Uno Code

```

// This code is the final version of code on the Uno. It responds to inputs from the Fio, sonar
// sensor, and hall effect sensor
// to control the RC car.
// This code works in conjunction with PitchRollHeading_Final on the Fio.
// Written by Jimmy Herring & Calvin Troxell
// Last edited: CT 12/16/14 4:21pm

```

```

// Included libraries
#include <Servo.h>

```

```

////////////////////////////////////Global Variables////////////////////////////////////

```

```

// Create servo objects
Servo myservo; // Create servo object to control a servo
Servo esc;     // Create servo object to control esc

```

```

// Set the initial steering and throttle values
int roll = 90; // Set initial roll to 90, this will ensure that steering starts forward
int pitch = 0; // Set initial pitch so that the car is not moving when first powered on

```

```

// Communication function variables
int p = 0; // Variable used with communicate function to get pitch from Fio
int r = 0; // Variable used with communicate function to get roll from Fio

```

```

// Sonar values
int pwPin = 7; // Pin used to read the sonar sensor
int sonar_pulse = 0; // Signal initially read from sonar sensor
int inches_present = 0; // The current distance that the car is from any object

```

```

// RPM functions variables
volatile byte half_revolutions; // Number of half revolutions detected by hall effect sensor
unsigned int rpm; // RPMs of back right wheel on RC car
unsigned int ft_s; // Speed of RC car in ft/s
unsigned long timeold; // Time when RPMs were last calculated

```

```

// Distance function variables

```

```

int obstacle_distance;          // Distance it will take for the car to brake and come to a complete
stop
float brake_constant = 0.05;    // Brake constant used in distance function to calculate distance
it will take car
                                // to come to a complete stop

// The following variables are for sonar mode filtering
int sonar_array_size = 3;      // Quantity of values used in mode calculation, must be an odd
number
int sonar_value[] = {0,0,0};   // An empty array to hold sonar sensor readings
int sonar_mode;                // Mode of sonar sensor readings

// The following variables are for rpm mode filtering
int rpm_array_size = 3;        // Quantity of values used in mode calculation, must be an odd
number
int rpm_value[] = {0,0,0};     // An empty array to hold rpm value readings
int rpm_mode;                  // Mode of rpm readings

// LED Variable
int brakeLED = 3;              // Pin that the brake lights are assigned to on the Uno

////////////////////////////////////Setup////////////////////////////////////

void setup()
{
    // Initialize Servos
    myservo.attach(9); // Attaches the servo on pin 9 to the servo object
    esc.attach(8);     // Attaches the esc on pin 8 to the servo object

    // Begin and configure serial monitor
    Serial.begin(9600); // Begins the serial monitor at a baud rate of 9600
    Serial.setTimeout(50); // Sets the maximum amount of time that the serial monitor will wait for
data to 50 milliseconds

    // Configure hall effect sensor
    attachInterrupt(0, rpm_init, RISING); // Use interrupt to measure every time pin goes from
low to high
    half_revolutions = 0;                  // Set initial half revolutions to zero
    rpm = 0;                              // Set initial RPMs to zero
    timeold = 0;                           // Set initial timeold to zero

    // Configure Brake LED
    pinMode(brakeLED, OUTPUT);             // Set brakeLED as an output

    // Configure the ESC so that it will respond to control from the Uno
    forward();

```

```

delayMicroseconds(100);
brake();
delayMicroseconds(100);
}

//////////////////////////////////Loop//////////////////////////////////

void loop()
{
  rpm_calc();    // Runs the rpm_calc function to calculate the RPMs of back right wheel on RC
car
  communicate(); // Runs the communicate function to bring data from the Fio
  sonar();       // Runs the sonar function to get the cars distance from any obstacles
  distance();    // Runs the distance function to calculate the minimum stopping distance to
avoid collision

  roll = map(r, -90, 90, 0, 180);    // Scales the roll from -90 to 90 to 0 to 179
  myservo.write(roll);               // Sets the servo position according to the scaled value
  pitch = map( p, -90, 90, 1350, 1650); // Scales the pitch from -90 to 90 to 1350 to 1650.
  // The ESC can handle a signal between 1000 and 2000. The speeds are being restricted in this
program.

  if (sonar_mode < 12){ // If the RC car is closer than 12 inches to an object,
    if (pitch > 1560){ // restrict the forward speed to a maximum of 1560
      pitch = 1560;
      drive();
    }
    else{
      drive();
    }
  }
  else{
    if (pitch > 1500){ // If the car is driving forward engage autobraking.
      if (sonar_mode > obstacle_distance){ // If the nearest object distance is greater than the
minimum brake distance,
        drive(); // allow the car to drive normally.
      }
      else{ // If the nearest object distance is less than the minimum brake
distance,
        if(Serial.available()>0){ // apply the brakes.
          Serial.read(); // Serial read while brakes are being applied to prevent data
from being stored
        } // in buffer.
        for (rpm_mode; rpm_mode > 1; rpm_mode--){ // Instead of using a set delay for the
brakes, this loop will brake

```

```

        digitalWrite(brakeLED,HIGH);          // based on the last RPM reading. Turn on brake
lights.
        brake();
        pitch = 1500;                        // Sets pitch to 1500 so car does not immediately drive
forward after
        delay(10);                          // autobraking.

        // Serial print for debugging purposes
        //Serial.println("braking");
    }
    rpm = 0;                                // Resets obstacle distance so that the car can drive.
    digitalWrite(brakeLED,LOW); // Turn off brake lights.
}
}
else{ // Allows the car to drive unimpeded in reverse. There is no autobraking for reverse.
    drive();
}
}
}

```

////////////////////////////////////Functions////////////////////////////////////

```

void communicate()
{
    // This function is used to different values from the Fio.
    // The variables sent by the Fio are pitch, p, and roll, r.
    // This function is based on code from Michael Robinson.

```

```

    char serialInput;

```

```

    if(Serial.available()>0);
    {
        serialInput = Serial.read();
        switch(serialInput)
        {
            case 'r':
                r=Serial.parseInt();
                break;

            case 'p':
                p=Serial.parseInt();
                break;

            default:
                break;
        }
    }

```

```

}

}

void sonar()
{
    // This function is used to control the sonar sensor.

    pinMode(pwPin, INPUT); // Sets the pin that will receive the sonar signal
    // Used to read in the pulse that is being sent by the MaxSonar device.
    // Pulse Width representation with a scale factor of 147 uS per Inch.

    sonar_pulse = pulseIn(pwPin, HIGH); // Reads the pulse value from the sonar sensor

    // Stores values from the sonar sensor in an array so that the values can be mode filtered.
    for(int i = 0; i < sonar_array_size; i++)
    {
        inches_present = sonar_pulse/147; // Converts the pulse to inches
        sonar_value[i] = inches_present;
    }

    sonar_mode = mode(sonar_value,sonar_array_size); // Mode of the sonar array is found

    // Serial print for debugging purposes
    //Serial.print("inches");
    //Serial.println(sonar_mode);

}

void drive()
{
    if (pitch > 1510){ // If the pitch is greater than 1510, the RC car will drive forward.
        forward();
    }
    else{
        if (pitch > 1491 && pitch < 1509) { // If the pitch is greater than 1491 and less than 1509,
            // the RC car will brake.

            brake();
        }
        else{
            reverse(); // If the pitch is less than 1491, the RC car will drive in reverse.
        }
    }
}

//Mode function, returning the mode or median.

```



```

int mode(int *x,int n){

    int i = 0;
    int count = 0;
    int maxCount = 0;
    int mode = 0;
    int bimodal;
    int prevCount = 0;
    while(i<(n-1)){
        prevCount=count;
        count=0;
        while(x[i]==x[i+1]){
            count++;
            i++;
        }
        if(count>prevCount&count>maxCount){
            mode=x[i];
            maxCount=count;
            bimodal=0;
        }
        if(count==0){
            i++;
        }
        if(count==maxCount){//If the dataset has 2 or more modes.
            bimodal=1;
        }
        if(mode==0||bimodal==1){//Return the median if there is no mode.
            mode=x[(n/2)];
        }
        return mode;
    }
}

void forward()
{
    esc.writeMicroseconds(pitch); // ESC writes the pitch value
    delayMicroseconds(100);      // Time for ESC to write pitch value
}

void brake()
{
    esc.writeMicroseconds(1500); // ESC writes 1500 so that the car brakes
    delayMicroseconds(100);      // Time for ESC to write brake value
}

void reverse()

```

```

{
  esc.writeMicroseconds(pitch); // ESC writes the pitch value
  delayMicroseconds(100);      // Time for ESC to write pitch value
}

void rpm_init()
{
  half_revolutions++;
  // Each rotation, the interrupt function is run twice
}

void rpm_calc(){
  if (half_revolutions >= 1) {
    // Update RPM every count

    rpm = 30*1000/(millis() - timeold)*half_revolutions; // Calculate the RPMs
    timeold = millis(); // Set timeold to current time
    half_revolutions = 0; // Reset half_revolutions to zero for next RPM calculation

    // Store RPM values in an array for mode filtering
    for(int i = 0; i < rpm_array_size; i++)
    {
      rpm_value[i] = rpm;
    }

    rpm_mode = mode(rpm_value,rpm_array_size); // Mode of the RPM array is found
    ft_s = (rpm_mode*3.5)/(720); // Speed of car is found using circumference of wheel
    multiplied by RPMs

    // Serial print for debugging purposes
    //Serial.print("rpm"); // for testing purposes
    //Serial.println(rpm_mode);
  }
}

void distance()
{
  obstacle_distance = (ft_s * ft_s / brake_constant * 12) + 6; // Calculates the minimum braking
  distance

  // Serial print for debugging purposes
  //Serial.print("crash"); // for testing purposes
  //Serial.println(obstacle_distance);
}

```

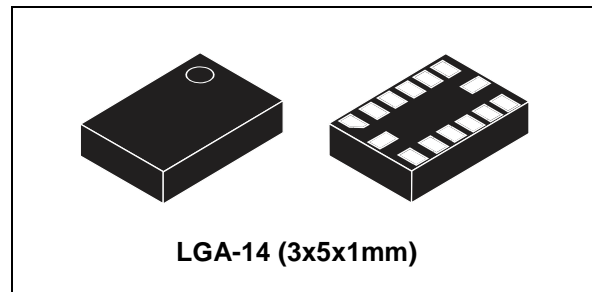


Ultra compact high performance e-compass 3D accelerometer and 3D magnetometer module

Preliminary data

Features

- 3 magnetic field channels and 3 acceleration channels
- From ± 1.3 to ± 8.1 gauss magnetic field full-scale
- $\pm 2g/\pm 4g/\pm 8g/\pm 16g$ selectable full-scale
- 16 bit data output



- I²C serial interface
- Analog supply voltage 2.16 V to 3.6 V
- Power-down mode/ low-power mode
- 2 independent programmable interrupt generators for free-fall and motion detection
- Embedded temperature sensor
- Embedded FIFO
- 6D/4D orientation detection
- ECOPACK[®] RoHS and “Green” compliant

Applications

- Compensated compass
- Map rotation
- Position detection
- Motion-activated functions
- Free-fall detection
- Click/double click recognition
- Pedometer
- Intelligent power-saving for handheld devices
- Display orientation
- Gaming and virtual reality input devices

- Vibration monitoring and compensation

Table 1. Device summary Description

The LSM303DLHC is a system-in-package featuring a 3D digital linear acceleration sensor and a 3D digital magnetic sensor. LSM303DLHC has linear acceleration full-scales of $\pm 2g$ / $\pm 4g$ / $\pm 8g$ / $\pm 16g$ and a magnetic field fullscale of ± 1.3 / ± 1.9 / ± 2.5 / ± 4.0 / ± 4.7 / ± 5.6 / ± 8.1 gauss. All full-scales available are fully selectable by the user.

LSM303DLHC includes an I²C serial bus interface that supports standard and fast mode 100 kHz and 400kHz. The system can be configured to generate interrupt signals by inertial wakeup/free-fall events as well as by the position of the device itself. Thresholds and timing of interrupt generators are programmable by the end user on the fly. Magnetic and accelerometer parts can be enabled or put into power-down mode separately.

The LSM303DLHC is available in a plastic land grid array package (LGA) and is guaranteed to operate over an extended temperature range from -40 °C to +85 °C.

April 2011

Doc ID 018771 Rev 1

1/42

This is preliminary information on a new product now in development or undergoing evaluation. Details are subject to change without notice.

www.st.com

- Impact recognition and logging

Contents

Part number	Temperature range [°C]	Package	Packing
LSM303DLHC	-40 to +85	LGA-14	Tray
LSM303DLHCTR	-40 to +85	LGA-14	Tape and reel

Contents

1	Block diagram and pin description	7
1.1	Block diagram	7
1.2	Pin description	8
2	Module specifications	9
2.1	Sensor characteristics	9

2.2	Temperature sensor characteristics	10
2.3	Electrical characteristics	11
2.4	Communication interface characteristics	12
2.4.1	Sensor I ² C - inter IC control interface	12
2.5	Absolute maximum ratings	13
2.6	Terminology	14
2.6.1	Linear acceleration sensitivity	14
2.6.2	Zero-g level	14
3	Functionality	15
3.1	Factory calibration	15
4	Application hints	16
4.1	External capacitors	16
4.2	Pull-up resistors	16
4.3	Digital interface power supply	17
4.4	Soldering information	17
4.5	High current wiring effects	17
5	Digital interfaces	18
5.1	I ² C serial interface	18
5.1.1	I ² C operation	19
5.1.2	Linear acceleration digital interface	20
5.1.3	Magnetic field digital interface	21
6	Register mapping	22
LSM303DLHC		Contents
7	Register description	24

7.1	Linear acceleration register description	24
7.1.1	CTRL_REG1_A (20h)	24
7.1.2	CTRL_REG2_A (21h)	25
7.1.3	CTRL_REG3_A (22h)	25
7.1.4	CTRL_REG4_A (23h)	26
7.1.5	CTRL_REG5_A (24h)	26
7.1.6	CTRL_REG6_A (25h)	27
7.1.7	REFERENCE/DATACAPTURE_A (26h)	27
7.1.8	STATUS_REG_A (27h)	28
7.1.9	OUT_X_L_A (28h), OUT_X_H_A (29h)	28
7.1.10	OUT_Y_L_A (2Ah), OUT_Y_H_A (2Bh)	28
7.1.11	OUT_Z_L_A (2Ch), OUT_Z_H_A (2Dh)	28
7.1.12	FIFO_CTRL_REG_A (2Eh)	29
7.1.13	FIFO_SRC_REG_A (2Fh)	29
7.1.14	INT1_CFG_A (30h)	29
7.1.15	INT1_SRC_A (31h)	30
7.1.16	INT1_THS_A (32h)	31
7.1.17	INT1_DURATION_A (33h)	31

7.1.18	INT2_CFG_A (34h)	31
7.1.19	INT2_SRC_A (35h)	32
7.1.20	INT2_THS_A (36h)	33
7.1.21	INT2_DURATION_A (37h)	33
7.1.22	CLICK_CFG_A (38h)	34
7.1.23	CLICK_SRC_A (39h)	34
7.1.24	CLICK_THS_A (3Ah)	35
7.1.25	TIME_LIMIT_A (3Bh)	35
7.1.26	TIME_LATENCY_A (3Ch)	35
7.1.27	TIME WINDOW_A (3Dh)	36
7.2	Magnetic field sensing register description	36
7.2.1	CRA_REG_M (00h)	36
7.2.2	CRB_REG_M (01h)	37
7.2.3	MR_REG_M (02h)	37
7.2.4	OUT_X_H_M (03), OUT_X_LH_M (04h)	38
7.2.5	OUT_Z_H_M (05), OUT_Z_L_M (06h)	38
7.2.6	OUT_Y_H_M (07), OUT_Y_L_M (08h)	38
7.2.7	SR_REG_M (09h)	38

Contents

	7.2.8	IR_REG_M (0Ah/0Bh/0Ch)	
		... 38	
	7.2.9	TEMP_OUT_H_M (31h), TEMP_OUT_L_M (32h)	
	 39	
8		Package information	40
9		Revision history	41

List of tables

Table 1.	Device summary.	1
Table 2.	Pin description	8
Table 3.	Sensor characteristics	9
Table 4.	Temperature sensor characteristics	10
Table 5.	Electrical characteristics	11
Table 6.	I ² C slave timing values	12
Table 7.	Absolute maximum ratings	13
Table 8.	Accelerometer operating mode selection	15
Table 9.	Serial interface pin description	18
Table 10.	Serial interface pin description	18
Table 11.	Transfer when master is writing one byte to slave	19
Table 12.	Transfer when master is writing multiple bytes to slave:	19
Table 13.	Transfer when master is receiving (reading) one byte of data from slave:	19
Table 14.	SAD+read/write patterns.	20
Table 15.	Transfer when master is receiving (reading) multiple bytes of data from slave	20
Table 16.	SAD	21
Table 17.	Register address map.	22
Table 18.	CTRL_REG1_A register	24
Table 19.	CTRL_REG1_A description	24

Table 20.	Data rate configuration	24
Table 21.	CTRL_REG2_A register	25
Table 22.	CTRL_REG2_A description	25
Table 23.	High pass filter mode configuration	25
Table 24.	CTRL_REG3_A register	25
Table 25.	CTRL_REG3_A description	26
Table 26.	CTRL_REG4_A register	26
Table 27.	CTRL_REG4_A description	26
Table 28.	CTRL_REG5_A register	26
Table 29.	CTRL_REG5_A description	27
Table 30.	CTRL_REG6_A register	27
Table 31.	CTRL_REG6_A description	27
Table 32.	REFERENCE_A register	27
Table 33.	REFERENCE_A register description	28
Table 34.	STATUS_A register	28
Table 35.	STATUS_A register description	28
Table 36.	REFERENCE_A register	29
Table 37.	REFERENCE_A register description	29
Table 38.	FIFO mode configuration	29
Table 39.	FIFO_SRC_A register.	29
Table 40.	INT1_CFG_A register.	29

Table 41.	INT1_CFG_A description	29
Table 42.	Interrupt mode	30
Table 43.	INT1_SRC_A register.	30
Table 44.	INT1_SRC_A description	30
Table 45.	INT1_THS_A register	31
Table 46.	INT1_THS_A description	31
Table 47.	INT1_DURATION_A register	31
Table 48.	INT1_DURATION_A description	31

List of tables

Table 49.	INT2_CFG_A register.	31
Table 50.	INT2_CFG_A description	32
Table 51.	Interrupt mode	32
Table 52.	INT2_SRC_A register.	32
Table 53.	INT2_SRC_A description	33
Table 54.	INT2_THS_A register	33
Table 55.	INT2_THS_A description	33
Table 56.	INT2_DURATION_A register	33
Table 57.	INT2_DURATION_A description	33
Table 58.	CLICK_CFG_A register	34
Table 59.	CLICK_CFG_A description.	34
Table 60.	CLICK_SRC_A register	34
Table 61.	CLICK_SRC_A description.	34

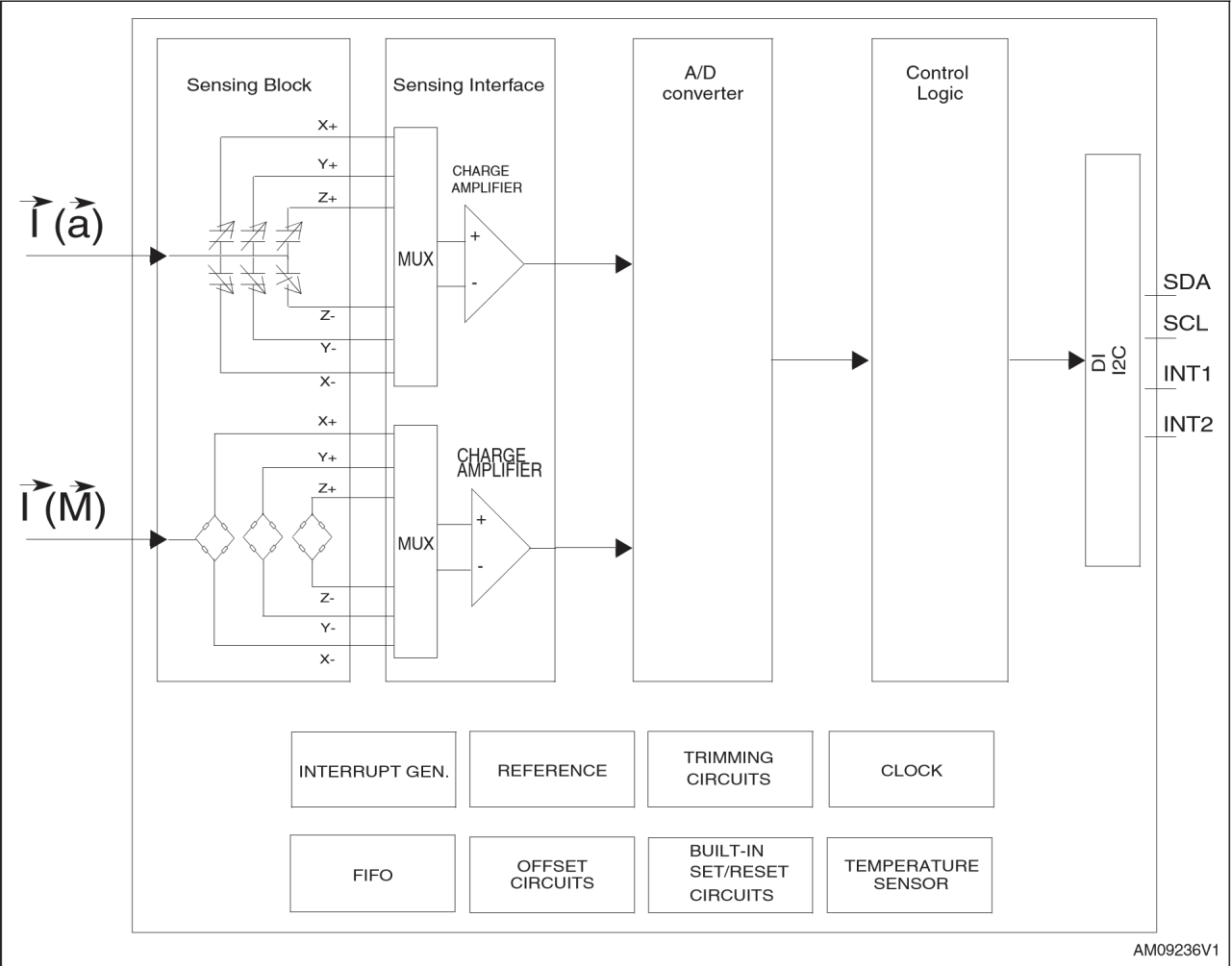
Table 62.	CLICK_THS_A register.	35
Table 63.	CLICK_SRC_A description.	35
Table 64.	TIME_LIMIT_A register.	35
Table 65.	TIME_LIMIT_A description	35
Table 66.	TIME_LATENCY_A register	35
Table 67.	TIME_LATENCY_A description	36
Table 68.	TIME_WINDOW_A register	36
Table 69.	TIME_WINDOW_A description.	36
Table 70.	CRA_REG_M register	36
Table 71.	CRA_REG_M description.	36
Table 72.	Data rate configurations	36
Table 73.	CRA_REG register	37
Table 74.	CRA_REG description	37
Table 75.	Gain setting.	37
Table 76.	MR_REG	37
Table 77.	MR_REG description	38
Table 79.	SR register	38
Table 80.	SR register description	38
Table 81.	IRA_REG_M.	38
Table 82.	IRB_REG_M.	38
Table 83.	IRC_REG_M.	39

Table 84.	TEMP_OUT_H_M register	39
Table 85.	TEMP_OUT_L_M register	39
Table 86.	TEMP_OUT resolution	39
Table 87.	Revision history	41

1 Block diagram and pin description

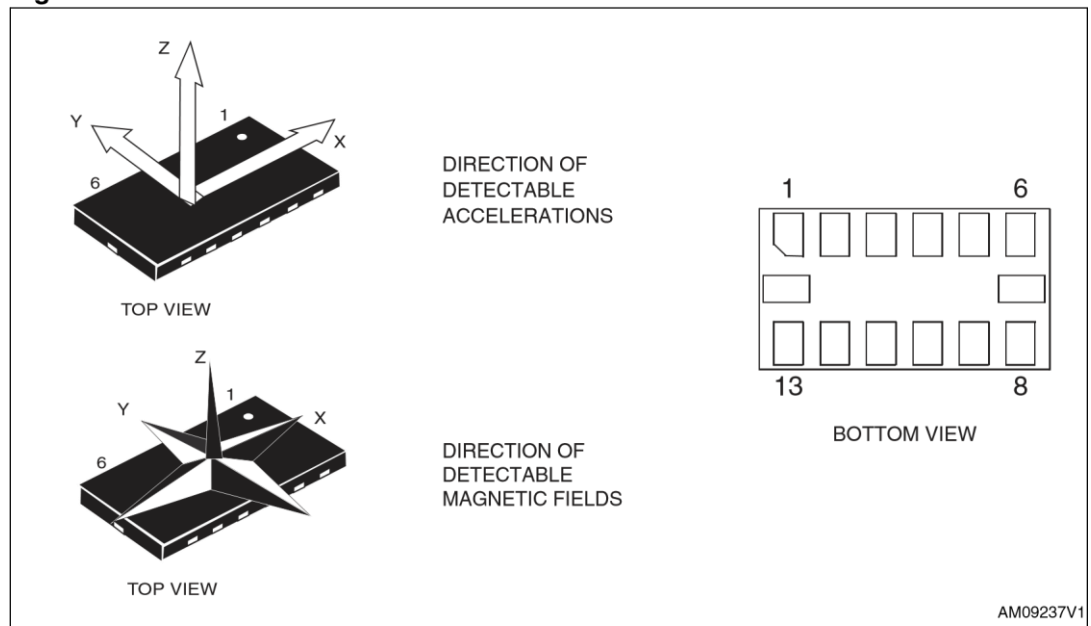
1.1 Block diagram

Figure 1. Block diagram



Block diagram and pin description

1.2 Pin description

Figure 2. Pin connection**Table 2. Pin description**

Pin#	Name	Function
1	Vdd_IO	Power supply for I/O pins
2	SCL	Signal interface I ² C serial clock (SCL)
3	SDA	Signal interface I ² C serial data (SDA)
4	INT2	Inertial Interrupt 2
5	INT1	Inertial Interrupt 1
6	C1	Reserved capacitor connection (C1)
7	GND	0 V supply
8	Reserved	Leave unconnected
9	DRDY	Data ready
10	Reserved	Connect to GND
11	Reserved	Connect to GND
12	SETP	S/R capacitor connection (C2)
13	SETC	S/R capacitor connection (C2)
14	Vdd	Power supply

2 Module specifications

2.1 Sensor characteristics

@ Vdd = 2.5 V, T = 25 °C unless otherwise noted^(a).

Table 3. Sensor characteristics

Symbol	Parameter	Test conditions	Min.	Typ. ⁽¹⁾	Max.	Unit
LA_FS	Linear acceleration measurement range ⁽²⁾	FS bit set to 00		±2		<i>g</i>
		FS bit set to 01		±4		
		FS bit set to 10		±8		
		FS bit set to 11		±16		
M_FS	Magnetic measurement range	GN bits set to 001		±1.3		gauss
		GN bits set to 010		±1.9		
		GN bits set to 011		±2.5		
		GN bits set to 100		±4.0		
		GN bits set to 101		±4.7		
		GN bits set to 110		±5.6		
		GN bits set to 111		±8.1		
LA_So	Linear acceleration sensitivity	FS bit set to 00		1		mg/LSB
		FS bit set to 01		2		
		FS bit set to 10		4		
		FS bit set to 11		12		
M_GN	Magnetic gain setting	GN bits set to 001 (X,Y)		1100		LSB/ gauss
		GN bits set to 001 (Z)		980		
		GN bits set to 010 (X,Y)		855		
		GN bits set to 010 (Z)		760		
		GN bits set to 011 (X,Y)		670		

	GN bits set to 011 (Z)		600	
	GN bits set to 100 (X,Y)		450	
	GN bits set to 100 (Z)		400	
	GN bits set to 101 (X,Y)		400	
	GN bits set to 101 (Z)		355	
	GN bits set to 110 (X,Y)		330	
	GN bits set to 110 (Z)		295	
	GN bits set to 111 ⁽²⁾ (X,Y)		230	
	GN bits set to 111 ⁽²⁾ (Z)		205	

- a. The product is factory calibrated at 2.5 V. The operational power supply range is from 2.16 V to 3.6 V.

Table 3. Sensor characteristics (continued)

Symbol	Parameter	Test conditions	Min.	Typ. ⁽¹⁾	Max.	Unit
LA_TCS ₀	Linear acceleration sensitivity change vs. temperature	FS bit set to 00		±0.01		%/°C
LA_TyOff	Linear acceleration typical Zero- <i>g</i> level offset accuracy ^{(3),(4)}	FS bit set to 00		±60		m <i>g</i>
LA_TCOff	Linear acceleration Zero- <i>g</i> level change vs. temperature	Max. delta from 25 °C		±0.5		m <i>g</i> /°C
LA_An	Acceleration noise density	FS bit set to 00, normal mode(Table 8.), ODR bit set to 1001		220		µg/sqrt(Hz)
M_R	Magnetic resolution			2		mgauss
M_CAS	Magnetic cross-axis sensitivity	Cross field = 0.5 gauss H applied = ±3 gauss		±1		%FS/ gauss
M_EF	Maximum exposed field	No permitting effect on zero reading			10000	gauss
M_DF	Disturbing field	Sensitivity starts to degrade. Use S/R pulse to restore sensitivity			20	gauss
Top	Operating temperature range		-40		+85	°C

1. Typical specifications are not guaranteed.

2. Verified by wafer level test and measurement of initial offset and sensitivity.

3. Typical Zero-*g* level offset value after MSL3 preconditioning.

4. Offset can be eliminated by enabling the built-in high pass filter.

2.2 Temperature sensor characteristics

@ V_{dd} = 2.5 V, T = 25 °C unless otherwise noted ^(b).

Table 4. Temperature sensor characteristics

Symbol	Parameter	Test condition	Min.	Typ. ⁽¹⁾	Max.	Unit
TSDr	Temperature sensor output change vs. temperature	-		8		LSB/°C ⁽²⁾
TODR	Temperature refresh rate			ODR ₍₃₎		Hz
Top	Operating temperature range		-40		+85	°C

1. Typical specifications are not guaranteed.



2. 12-bit resolution.
3. For ODR configuration refer to [Table 72](#).

b. The product is factory calibrated at 2.5 V.

2.3 Electrical characteristics

@ Vdd = 2.5 V, T = 25 °C unless otherwise noted.

Table 5. Electrical characteristics

Symbol	Parameter	Test conditions	Min.	Typ. ⁽¹⁾	Max.	Unit
Vdd	Supply voltage	-	2.16		3.6	V
Vdd_IO	Module power supply for I/O		1.71	1.8	Vdd+0.1	
Idd	Current consumption in normal mode ⁽²⁾			110		μA
IddSL	Current consumption in sleep-mode ⁽³⁾			1		μA
Top	Operating temperature range		-40		+85	°C

1. Typical specifications are not guaranteed.
2. Magnetic sensor setting ODR = 7.5 Hz, Accelerometer sensor ODR = 50 Hz.
3. Linear accelerometer in sleep-mode and magnetic sensor in power-down mode.

2.4 Communication interfaces characteristics

External pull-up resistors are required to support I²C standard and fast speed modes.

2.4.1 Sensor I²C - inter IC control interface

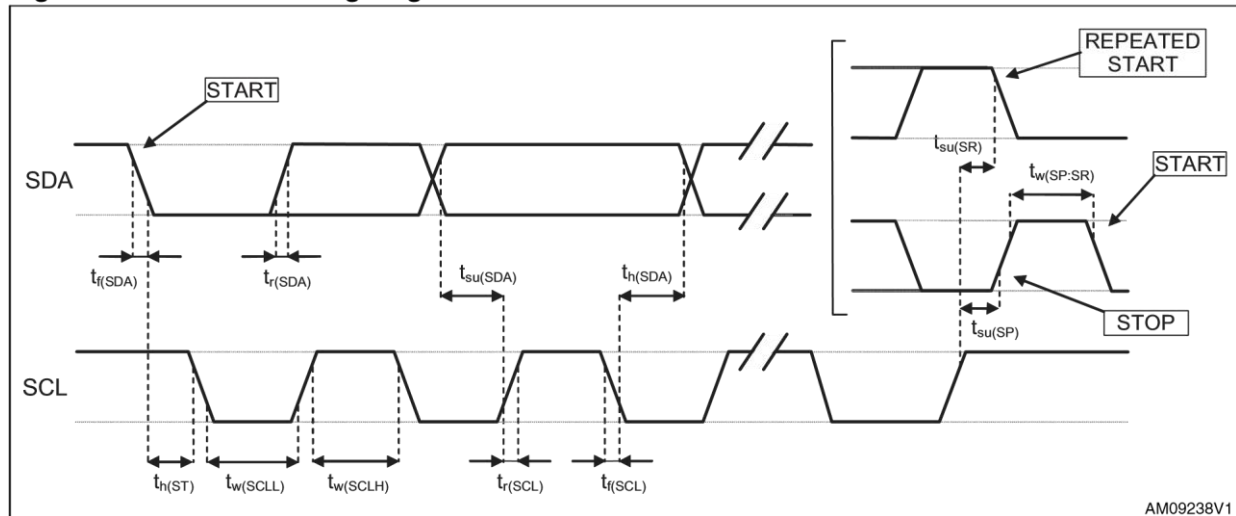
Subject to general operating conditions for Vdd and Top.

Table 6. I²C slave timing values

Symbol	Parameter	I ² C standard mode ⁽¹⁾		I ² C fast mode ⁽¹⁾		Unit
		Min.	Max.	Min.	Max.	
f(SCL)	SCL clock frequency	0	100	0	400	KHz
tw(SCLL)	SCL clock low time	4.7		1.3		μs
tw(SCLH)	SCL clock high time	4.0		0.6		
tsu(SDA)	SDA setup time	250		100		ns
th(SDA)	SDA data hold time	0.01	3.45	0.01	0.9	μs
tr(SDA) t _r (SCL)	SDA and SCL rise time		1000	20 + 0.1Cb ⁽²⁾	300	ns
tf(SDA) t _f (SCL)	SDA and SCL fall time		300	20 + 0.1Cb ⁽²⁾	300	
th(ST)	START condition hold time	4		0.6		μs
tsu(SR)	Repeated START condition setup time	4.7		0.6		
tsu(SP)	STOP condition setup time	4		0.6		
tw(SP:SR)	Bus free time between STOP and START condition	4.7		1.3		

1. Data based on standard I²C protocol requirement, not tested in production.

2. Cb = total capacitance of one bus line, in pF.

Figure 3. I²C slave timing diagram (c)

2.5 Absolute maximum ratings

Stresses above those listed as “absolute maximum ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device under these conditions is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

Table 7. Absolute maximum ratings

Symbol	Ratings	Maximum value	Unit
Vdd	Supply voltage	-0.3 to 4.8	V
Vdd_IO	I/O pins supply voltage	-0.3 to 4.8	V
Vin	Input voltage on any control pin (SCL, SDA)	-0.3 to Vdd_IO +0.3	V
APOW	Acceleration (any axis, powered, Vdd = 2.5 V)	3,000 for 0.5 ms	g
		10,000 for 0.1 ms	g
AUNP	Acceleration (any axis, unpowered)	3,000 for 0.5 ms	g
		10,000 for 0.1 ms	g
TOP	Operating temperature range	-40 to +85	°C
TSTG	Storage temperature range	-40 to +125	°C



This is a mechanical shock sensitive device, improper handling can cause permanent damage to the part.



This is an ESD sensitive device, improper handling can cause permanent damage to the part.

-
- c. Measurement points are done at $0.2 \cdot V_{dd_IO}$ and $0.8 \cdot V_{dd_IO}$, for both ports.

2.6 Terminology

2.6.1 Linear acceleration sensitivity

Linear acceleration sensitivity describes the gain of the accelerometer sensor and can be determined by applying 1 g acceleration to it. As the sensor can measure DC accelerations, this can be done easily by pointing the axis of interest towards the center of the Earth, noting the output value, rotating the sensor by 180 degrees (pointing to the sky) and noting the output value again. By doing so, $\pm 1 g$ acceleration is applied to the sensor. Subtracting the larger output value from the smaller one, and dividing the result by 2, leads to the actual sensitivity of the sensor. This value changes very little over temperature and also very little over time. The sensitivity tolerance describes the range of sensitivities of a large population of sensors.

2.6.2 Zero-g level

Zero- g level offset (TyOff) describes the deviation of an actual output signal from the ideal output signal if no acceleration is present. A sensor in a steady-state on a horizontal surface measures 0 g in the X axis and 0 g in the Y axis whereas the Z axis measures 1 g . The output is ideally in the middle of the dynamic range of the sensor (content of OUT registers 00h, data expressed as 2's complement number). A deviation from the ideal value in this case is called Zero- g offset. Offset is, to some extent, a result of stress to the MEMS sensor and therefore the offset can slightly change after mounting the sensor onto a printed circuit board or exposing it to extensive mechanical stress. Offset changes little over temperature, see "Zero- g level change vs. temperature". The Zero- g level tolerance (TyOff) describes the standard deviation of the range of Zero- g levels of a population of sensors.

3 Functionality

The LSM303DLHC is a system-in-package featuring a 3D digital linear acceleration and 3D digital magnetic field detection sensor.

The system includes specific sensing elements and an IC interface capable of measuring both the linear acceleration and magnetic field applied on it and to provide a signal to the external world through an I²C serial interface with separated digital output.

The sensing system is manufactured using specialized micromachining processes, while the IC interfaces are realized using a CMOS technology that allows to design a dedicated circuit which is trimmed to better match the sensing element characteristics.

The LSM303DLHC features two data-ready signals (RDY) which indicate when a new set of measured acceleration data and magnetic data are available, therefore simplifying data synchronization in the digital system that uses the device.

The LSM303DLHC may also be configured to generate a free-fall interrupt signal according to a programmed acceleration event along the enabled axes.

Linear acceleration operating mode

LSM303DLHC provides two different acceleration operating modes, respectively reported as “normal mode” and “low-power mode”. While normal mode guarantees high resolution, low-power mode reduces further the current consumption.

[Table 8](#) summarizes how to select the operating mode.

Table 8. Accelerometer operating mode selection

Operating mode	CTRL_REG1[3] (LPen bit)	CTRL_REG4[3] (HR bit)	BW [Hz]	Turn-on time [ms]
Low-power mode	1	0	ODR/2	1
Normal mode	0	1	ODR/9	7/ODR

3.1 Factory calibration

The IC interface is factory calibrated for linear acceleration sensitivity (LA_{So}), and linear acceleration Zero-g level (LA_{TyOff}).

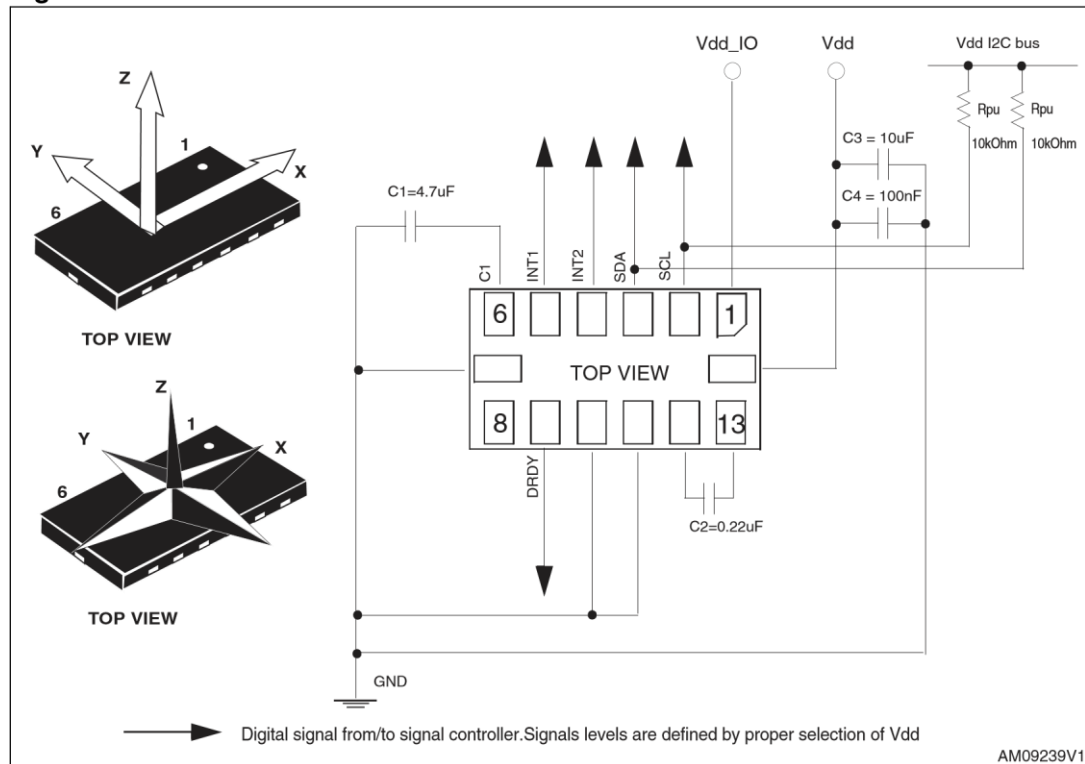
The trimming values are stored inside the device by a non-volatile memory. Any time the device is turned on, the trimming parameters are downloaded into the registers to be used during the normal operation. This allows the user to use the device without further calibration.

Application hints



4 Application hints

Figure 4. LSM303DLHC electrical connection



4.1 capacitors

The C1 and C2 external capacitors should be low SR value ceramic type constructions (typ. suggested value 200 mOhm). Reservoir capacitor C1 is nominally 4.7 μF in capacitance, with the set/reset capacitor C2 nominally 0.22 μF in capacitance.

The device core is supplied through the Vdd line. Power supply decoupling capacitors ($C4 = 100$ nF ceramic, $C3 = 10$ μF Al) should be placed as near as possible to the supply pin of the device (common design practice). All the voltage and ground supplies must be present at the same time to have proper behavior of the IC (refer to [Figure 4](#)).

The functionality of the device and the measured acceleration/magnetic field data is selectable and accessible through the I²C interface.

The functions, the threshold, and the timing of the two interrupt pins (INT 1 and INT 2) can be completely programmed by the user through the I²C interface.

4.2 Pull-up resistors

Pull-up resistors (suggested value 10 kOhm) are placed on the two I²C bus lines.

4.3 Digital interface power supply

This digital interface, dedicated to the linear acceleration and to the magnetic field signal, is capable of operating with a standard power supply (Vdd) or using a dedicated power supply (Vdd_IO).

4.4 Soldering information

The LGA package is compliant with the ECOPACK[®], RoHS, and “Green” standard. It is qualified for soldering heat resistance according to JEDEC J-STD-020.

Leave “Pin 1 Indicator” unconnected during soldering.

Land pattern and soldering recommendations are available at www.st.com/mems.

4.5 High current wiring effects

High current in the wiring and printed circuit trace can be culprits in causing errors in magnetic field measurements for compassing.

Conductor generated magnetic fields add to the Earth’s magnetic field, causing errors in compass heading computation.

Keep currents higher than 10 mA a few millimeters further away from the sensor IC.

Digital interfaces

5 Digital interfaces

The registers embedded inside the LSM303DLHC are accessible through two separate I²C serial interfaces, one for the accelerometer core and one for the magnetometer core.

Table 9. Serial interface pin description

PIN Name	PIN Description
SCL	I ² C serial clock (SCL)
SDA	I ² C serial data (SDA)

5.1 I²C serial interface

The LSM303DLHC I²C is a bus slave. The I²C is employed to write the data into the registers when also be read back.

The relevant I²C terminology is given in the table below.

Table 10. Serial interface pin description

Term	Description
Transmitter	The device which sends data to the bus
Receiver	The device which receives data from the bus
Master	The device which initiates a transfer, generates clock signals, and terminates a transfer
Slave	The device addressed by the master

There are two signals associated with the I²C bus, the serial clock line (SCL) and the serial data line (SDA). The latter is a bidirectional line used for sending and receiving the data to/from the interface.

Digital interfaces

5.1.1 I²C operation

The transaction on the bus is started through a START (ST) signal. A START condition is defined as a HIGH to LOW transition on the data line while the SCL line is held HIGH. After this has been transmitted by the master, the bus is considered busy. The next byte of data transmitted after the start condition contains the address of the slave in the first 7 bits and bit 8 tells whether the master is receiving data from the slave or transmitting data to the slave. When an address is

sent, each device in the system compares the first seven bits after a start condition with its address. If they match, the device considers itself addressed by the master. Data transfer with acknowledge is mandatory. The transmitter must release the SDA line during the acknowledge pulse. The receiver must then pull the data line LOW so that it remains stable low during the HIGH period of the acknowledge clock pulse. A receiver which has been addressed is obliged to generate an acknowledge after each byte of data received.

The I²C embedded inside the LSM303DLHC behaves like a slave device and the following protocol must be adhered to. After the start condition (ST) a slave address is sent, once a slave acknowledge (SAK) has been returned, an 8-bit sub-address (SUB) is transmitted; the 7 LSBs represent the actual register address while the MSB enables address autoincrement. If the MSB of the SUB field is '1', the SUB (register address) is automatically increased to allow multiple data Read/Write.

Table 11. Transfer when master is writing one byte to slave

Master	ST	SAD + W		SUB		DATA		SP
Slave			SAK		SAK		SAK	

Table

12. Transfer when master is writing multiple bytes to slave:

Master	ST	SAD + W		SUB		DATA		DATA		SP
Slave			SAK		SAK		SAK		SAK	

Table 13. Transfer when master is receiving (reading) one byte of data from slave:

Master	ST	SAD + W		SUB		SR	SAD + R			NMAK	SP
Slave			SAK		SAK			SAK	DATA		

Data are transmitted in byte format (DATA). Each data transfer contains 8 bits. The number of bytes transferred per transfer is unlimited. Data is transferred with the most significant bit (MSB) first. If a receiver can't receive another complete byte of data until it has performed some other function, it can hold the clock line SCL LOW to force the transmitter into a wait state. Data transfer only continues when the receiver is ready for another byte and releases the data line. If a slave receiver doesn't acknowledge the slave address (i.e. it is not able to receive because it is performing some real-time function) the data line must be left HIGH by the slave.

The master can then abort the transfer. A LOW to HIGH transition on the SDA line while the SCL line is HIGH is defined as a STOP condition. Each data transfer must be terminated by the generation of a STOP (SP) condition.

Digital interfaces

5.1.2 Linear acceleration digital interface

For linear acceleration the default (factory) 7-bit slave address is 0011001b.

The slave address is completed with a Read/Write bit. If the bit is '1' (read), a repeated START (SR) condition must be issued after the two sub-address bytes; if the bit is '0' (write) the master transmits to the slave with the direction unchanged. [Table 14](#) explains how the read/write bit pattern is composed, listing all the possible configurations.

Table 14.SAD+Read/Write patterns

Command	SAD[7:1]	R/W	SAD+R/W
Read	0011001	1	00110011 (33h)
Write	0011001	0	00110010 (32h)

In order to read multiple bytes, it is necessary to assert the most significant bit of the subaddress field. In other words, SUB(7) must be equal to 1 while SUB(6-0) represents the address of the first register to be read.

In the presented communication format, MAK is master acknowledge and NMAK is no master acknowledge.

Table 15.Transfer when master is receiving (reading) multiple bytes of data from slave

Master	ST	SAD +W		SUB		SR	SAD +R			MAK		MAK		NMAK	SP
Slave			SAK		SAK			SAK	DATA		DATA		DATA		

Digital interfaces

5.1.3 Magnetic field digital interface

For magnetic sensors the default (factory) 7-bit slave address is 0011110xb.

The slave address is completed with a Read/Write bit. If the bit is '1' (read), a repeated START (SR) condition must be issued after the two sub-address bytes; if the bit is '0' (write) the master transmits to the slave with the direction unchanged. [Table 16](#) explains how the SAD is composed.

Table 16. SAD

Command	SAD[6:0]	R/W	SAD+R/W
Read	0011110	1	00111101 (3Dh)
Write	0011110	0	00111100 (3Ch)

Magnetic signal interface reading/writing

The interface uses an address pointer to indicate which register location is to be read from or written to. These pointer locations are sent from the master to this slave device and succeed the 7-bit address plus 1 bit Read/Write identifier. To minimize the communication between the master and magnetic digital interface of LSM303DLHC, the address pointer updates automatically without master intervention.

This automatic address pointer update has two additional features. First, when address 12 or higher is accessed, the pointer updates to address 00, and secondly, when address 08 is reached, the pointer rolls back to address 03. Logically, the address pointer operation functions as shown below.

If (address pointer = 08) then the address pointer = 03

Or else, if (address pointer >= 12) then the address pointer = 0

Or else, (address pointer) = (address pointer) + 1

The address pointer value itself cannot be read via the I²C bus.

Any attempt to read an invalid address location returns 0, and any write to an invalid address location, or an undefined bit within a valid address location, is ignored by this device.

Register mapping**6 Register mapping**

[Table 17](#) provides a listing of the 8-bit registers embedded in the device and the related addresses:

Table 17. Register address map

Name	Slave address	Type	Register address		Default	Comment
			Hex	Binary		
Reserved (do not modify)	Table 14		00 - 1F	--	--	Reserved
CTRL_REG1_A	Table 14	rw	20	010 0000	00000111	
CTRL_REG2_A	Table 14	rw	21	010 0001	00000000	

LSM303DLHC

CTRL_REG3_A	Table 14	rw	22	010 0010	00000000	
CTRL_REG4_A	Table 14	rw	23	010 0011	00000000	
CTRL_REG5_A	Table 14	rw	24	010 0100	00000000	
CTRL_REG6_A	Table 14	rw	25	010 0101	00000000	
REFERENCE_A	Table 14	rw	26	010 0110	00000000	
STATUS_REG_A	Table 14	r	27	010 0111	00000000	
OUT_X_L_A	Table 14	r	28	010 1000	output	
OUT_X_H_A	Table 14	r	29	010 1001	output	
OUT_Y_L_A	Table 14	r	2A	010 1010	output	
OUT_Y_H_A	Table 14	r	2B	010 1011	output	
OUT_Z_L_A	Table 14	r	2C	010 1100	output	
OUT_Z_H_A	Table 14	r	2D	010 1101	output	
FIFO_CTRL_REG_A	Table 14	rw	2E	010 1110	00000000	
FIFO_SRC_REG_A	Table 14	r	2F	010 1111		
INT1_CFG_A	Table 14	rw	30	011 0000	00000000	
INT1_SOURCE_A	Table 14	r	31	011 0001	00000000	
INT1_THS_A	Table 14	rw	32	011 0010	00000000	
INT1_DURATION_A	Table 14	rw	33	011 0011	00000000	
INT2_CFG_A	Table 14	rw	34	011 0100	00000000	
INT2_SOURCE_A	Table 14	r	35	011 0101	00000000	
INT2_THS_A	Table 14	rw	36	011 0110	00000000	
INT2_DURATION_A	Table 14	rw	37	011 0111	00000000	
CLICK_CFG_A	Table 14	rw	38	011 1000	00000000	

CLICK_SRC_A	Table 14	rw	39	011 1001	00000000	
CLICK_THS_A	Table 14	rw	3A	011 1010	00000000	
TIME_LIMIT_A	Table 14	rw	3B	011 1011	00000000	

Register mapping
Table 17. Register address map (continued)

Name	Slave address	Type	Register address		Default	Comment
			Hex	Binary		
TIME_LATENCY_A	Table 14	rw	3C	011 1100	00000000	
TIME_WINDOW_A	Table 14	rw	3D	011 1101	00000000	
Reserved (do not modify)	Table 14		3E-3F	--	--	Reserved
CRA_REG_M	Table 16	rw	00	00000000	0001000	
CRB_REG_M	Table 16	rw	01	00000001	0010000	
MR_REG_M	Table 16	rw	02	00000010	00000011	
OUT_X_H_M	Table 16	r	03	00000011	output	
OUT_X_L_M	Table 16	r	04	00000100	output	
OUT_Z_H_M	Table 16	r	05	00000101	output	
OUT_Z_L_M	Table 16	r	06	00000110	output	
OUT_Y_H_M	Table 16	r	07	00000111	output	
OUT_Y_L_M	Table 16	r	08	00001000	output	
SR_REG_Mg	Table 16	r	09	00001001	00000000	
IRA_REG_M	Table 16	r	0A	00001010	01001000	
IRB_REG_M	Table 16	r	0B	00001011	00110100	
IRC_REG_M	Table 16	r	0C	00001100	00110011	

LSM303DLHC

Reserved (do not modify)	Table 16		0D-30	--	--	Reserved
TEMP_OUT_H_M	Table 16		31	00000000	output	
TEMP_OUT_L_M	Table 16		32	00000000	output	
Reserved (do not modify)	Table 16		33-3A	--	--	Reserved

Registers marked as “reserved” must not be changed. The writing to these registers may cause permanent damage to the device.

The content of the registers that are loaded at boot should not be changed. They contain the factory calibrated values. Their content is automatically restored when the device is powered up.

7 Register description

The device contains a set of registers which are used to control its behavior and to retrieve acceleration data. The register address, made up of 7 bits, is used to identify them and to write the data through the serial interface.

7.1 Linear acceleration register description

7.1.1 CTRL_REG1_A (20h)

Table 18. CTRL_REG1_A register

ODR3	ODR2	ODR1	ODR0	LPen	Zen	Yen	Xen
------	------	------	------	------	-----	-----	-----

Table 19. CTRL_REG1_A description

ODR3-0	Data rate selection. Default value: 0 (0000: power-down, others: refer to Table 20 .)
LPen	Low-power mode enable. Default value: 0 (0: normal mode, 1: low-power mode)
Zen	Z axis enable. Default value: 1 (0: Z axis disabled, 1: Z axis enabled)
Yen	Y axis enable. Default value: 1 (0: Y axis disabled, 1: Y axis enabled)
Xen	X axis enable. Default value: 1 (0: X axis disabled, 1: X axis enabled)

ODR<3:0> is used to set the power mode and ODR selection. In [Table 20](#) all frequencies resulting in a combination of ODR<3:0> are listed.

Table 20. Data rate configuration

ODR3	ODR2	ODR1	ODR0	Power mode selection
0	0	0	0	Power-down mode
0	0	0	1	Normal / low-power mode (1 Hz)
0	0	1	0	Normal / low-power mode (10 Hz)
0	0	1	1	Normal / low-power mode (25 Hz)
0	1	0	0	Normal / low-power mode (50 Hz)
0	1	0	1	Normal / low-power mode (100 Hz)

0	1	1	0	Normal / low-power mode (200 Hz)
0	1	1	1	Normal / low-power mode (400 Hz)

Table 20. Data rate configuration (continued)

ODR3	ODR2	ODR1	ODR0	Power mode selection
1	0	0	0	Low-power mode (1.620 KHz)
1	0	0	1	Normal (1.344 kHz) / low-power mode (5.376 KHz)

7.1.2**CTRL_REG2_A (21h)****Table 21. CTRL_REG2_A register**

HPM1	HPM0	HPCF2	HPCF1	FDS	HPCLICK	HPIS2	HPIS1
------	------	-------	-------	-----	---------	-------	-------

Table 22. CTRL_REG2_A description

HPM1 -HPM0	High pass filter mode selection. Default value: 00 (refer to Table 23)
HPCF2 - HPCF1	High pass filter cut-off frequency selection
FDS	Filtered data selection. Default value: 0 (0: internal filter bypassed, 1: data from internal filter sent to output register and FIFO)
HPCLICK	High pass filter enabled for CLICK function. (0: filter bypassed, 1: filter enabled)
HPIS2	High pass filter enabled for AOI function on Interrupt 2, (0: filter bypassed, 1: filter enabled)
HPIS1	High pass filter enabled for AOI function on Interrupt 1, (0: filter bypassed, 1: filter enabled)

Table 23. High pass filter mode configuration

HPM1	HPM0	High pass filter mode
0	0	Normal mode (reset reading HP_RESET_FILTER)
0	1	Reference signal for filtering
1	0	Normal mode
1	1	Autoreset on interrupt event

7.1.3**CTRL_REG3_A (22h)**

Table 24. CTRL_REG3_A register

I1_CLICK	I1_AOI1	I1_AOI2	I1_DRDY1	I1_DRDY2	I1_WTM	I1_OVERRUN	--
----------	---------	---------	----------	----------	--------	------------	----

Table 25. CTRL_REG3_A description

I1_CLICK	CLICK interrupt on INT1. Default value 0. (0: disable, 1: enable)
I1_AOI1	AOI1 interrupt on INT1. Default value 0. (0: disable, 1: enable)
I1_AOI2	AOI2 interrupt on INT1. Default value 0. (0: disable, 1: enable)
I1_DRDY1	DRDY1 interrupt on INT1. Default value 0. (0: disable, 1: enable)
I1_DRDY2	DRDY2 interrupt on INT1. Default value 0. (0: disable, 1: enable)
I1_WTM	FIFO watermark interrupt on INT1. Default value 0. (0: disable, 1: enable)
I1_OVERRUN	FIFO overrun interrupt on INT1. Default value 0. (0: disable, 1: enable)

7.1.4 CTRL_REG4_A (23h)

Table 26. CTRL_REG4_A register

BDU	BLE	FS1	FS0	HR	0(1)	0(1)	SIM
-----	-----	-----	-----	----	------	------	-----

1. This bit must be set to '0' for correct working of the device.

Table 27. CTRL_REG4_A description

BDU	Block data update. Default value: 0 (0: continuous update, 1: output registers not updated until MSB and LSB reading)
BLE	Big/little endian data selection. Default value 0. (0: data LSB @ lower address, 1: data MSB @ lower address)
FS1-FS0	Full-scale selection. Default value: 00 (00: +/- 2G, 01: +/- 4G, 10: +/- 8G, 11: +/- 16G)
HR	High resolution output mode: Default value: 0 (0: high resolution disable, 1: high resolution enable)
SIM	SPI serial interface mode selection. Default value: 0 (0: 4-wire interface, 1: 3-wire interface).

7.1.5 CTRL_REG5_A (24h)

Table 28. CTRL_REG5_A register

BOOT	FIFO_EN	--	--	LIR_INT1	D4D_INT1	LIR_INT2	D4D_INT2
------	---------	----	----	----------	----------	----------	----------

Table 29. CTRL_REG5_A description

BOOT	Reboot memory content. Default value: 0 (0: normal mode, 1: reboot memory content)
FIFO_EN	FIFO enable. Default value: 0 (0: FIFO disable, 1: FIFO enable)
LIR_INT1	Latch interrupt request on INT1_SRC register, with INT1_SRC register cleared by reading INT1_SRC itself. Default value: 0. (0: interrupt request not latched, 1: interrupt request latched)
D4D_INT1	4D enable: 4D detection is enabled on INT1 when 6D bit on INT1_CFG is set to 1.
LIR_INT2	Latch interrupt request on INT2_SRC register, with INT2_SRC register cleared by reading INT2_SRC itself. Default value: 0. (0: interrupt request not latched, 1: interrupt request latched)
D4D_INT2	4D enable: 4D detection is enabled on INT2 when 6D bit on INT2_CFG is set to 1.

7.1.6**CTRL_REG6_A (25h)****Table 30. CTRL_REG6_A register**

I2_CLICKen	I2_INT1	I2_INT2	BOOT_I1	P2_ACT	--	H_LACTIVE	-
------------	---------	---------	---------	--------	----	-----------	---

Table 31. CTRL_REG6_A description

I2_CLICKen	CLICK interrupt on PAD2. Default value 0. (0: disable, 1: enable)
I2_INT1	Interrupt 1 on PAD2. Default value 0. (0: disable, 1: enable)
I2_INT2	Interrupt 2 on PAD2. Default value 0. (0: disable, 1: enable)
BOOT_I1	Reboot memory content on PAD2. Default value: 0 (0: disable, 1: enable)
P2_ACT	Active function status on PAD2. Default value 0. (0: disable, 1: enable)
H_LACTIVE	Interrupt active high, low. Default value 0. (0: active high, 1: active low)

7.1.7**REFERENCE/DATACAPTURE_A (26h)**

Table 32.REFERENCE_A register

Ref7	Ref6	Ref5	Ref4	Ref3	Ref2	Ref1	Ref0
------	------	------	------	------	------	------	------

Table 33.REFERENCE_A register description

Ref 7-Ref0	Reference value for interrupt generation. Default value: 0
------------	--

7.1.8**STATUS_REG_A (27h)****Table 34.STATUS_A register**

ZYXOR	ZOR	YOR	XOR	ZYXDA	ZDA	YDA	XDA
-------	-----	-----	-----	-------	-----	-----	-----

Table 35.STATUS_A register description

ZYXOR	X, Y, and Z axis data overrun. Default value: 0 (0: no overrun has occurred, 1: a new set of data has overwritten the previous ones)
ZOR	Z axis data overrun. Default value: 0 (0: no overrun has occurred, 1: a new data for the Z-axis has overwritten the previous one)
YOR	Y axis data overrun. Default value: 0 (0: no overrun has occurred, 1: a new data for the Y-axis has overwritten the previous one)
XOR	X axis data overrun. Default value: 0 (0: no overrun has occurred, 1: a new data for the X-axis has overwritten the previous one)
ZYXDA	X, Y, and Z axis new data available. Default value: 0 (0: a new set of data is not yet available, 1: a new set of data is available)
ZDA	Z axis new data available. Default value: 0 (0: a new data for the Z-axis is not yet available, 1: a new data for the Z-axis is available)
YDA	Y axis new data available. Default value: 0 (0: a new data for the Y-axis is not yet available, 1: a new data for the Y-axis is available)

XDA	X axis new data available. Default value: 0 (0: a new data for the X-axis is not yet available, 1: a new data for the X-axis is available)
-----	--

7.1.9 OUT_X_L_A (28h), OUT_X_H_A (29h)

X-axis acceleration data. The value is expressed in 2's complement.

7.1.10 OUT_Y_L_A (2Ah), OUT_Y_H_A (2Bh)

Y-axis acceleration data. The value is expressed in 2's complement.

7.1.11 OUT_Z_L_A (2Ch), OUT_Z_H_A (2Dh)

Z-axis acceleration data. The value is expressed in 2's complement.

7.1.12 FIFO_CTRL_REG_A (2Eh)

Table 36.REFERENCE_A register

FM1	FM0	TR	FTH4	FTH3	FTH2	FTH1	FTH0
-----	-----	----	------	------	------	------	------

Table 37. REFERENCE_A register description

FM1-FM0	FIFO mode selection. Default value: 00 (see Table 38)
TR	Trigger selection. Default value: 0 0: trigger event linked to trigger signal on INT1 1: trigger event linked to trigger signal on INT2
FTH4:0	Default value: 0

Table 38. FIFO mode configuration

FM1	FM0	FIFO mode configuration
0	0	Bypass mode
0	1	FIFO mode
1	0	Stream mode
1	1	Trigger mode

7.1.13 FIFO_SRC_REG_A (2Fh)

Table 39.FIFO_SRC_A register

7.1.14

WTM	OVFN_FIFO	EMPTY	FSS4	FSS3	FSS2	FSS1	FSS0
-----	-----------	-------	------	------	------	------	------

INT1_CFG_A (30h)**Table 40. INT1_CFG_A register**

AOI	6D	ZHIE/ ZUPE	ZLIE/ ZDOWNE	YHIE/ YUPE	YLIE/ YDOWNE	XHIE/ XUPE	XLIE/ XDOWNE
-----	----	---------------	-----------------	---------------	-----------------	---------------	-----------------

Table 41. INT1_CFG_A description

AOI	AND/OR combination of interrupt events. Default value: 0 (refer to Table 42)
6D	6-direction detection function enabled. Default value: 0 (refer to Table 42)
ZHIE/ ZUPE	Enable interrupt generation on Z high event or on direction recognition. Default value: 0 (0: disable interrupt request, 1: enable interrupt request)
ZLIE/ ZDOWNE	Enable interrupt generation on Z low event or on direction recognition. Default value: 0 (0: disable interrupt request, 1: enable interrupt request)

Table 41. INT1_CFG_A description (continued)

YHIE/ YUPE	Enable interrupt generation on Y high event or on direction recognition. Default value: 0 (0: disable interrupt request, 1: enable interrupt request.)
YLIE/ YDOWNE	Enable interrupt generation on Y low event or on direction recognition. Default value: 0 (0: disable interrupt request, 1: enable interrupt request.)
XHIE/ XUPE	Enable interrupt generation on X high event or on direction recognition. Default value: 0 (0: disable interrupt request, 1: enable interrupt request.)
XLIE/XDOWNE	Enable interrupt generation on X low event or on direction recognition. Default value: 0 (0: disable interrupt request, 1: enable interrupt request.)

Content of this register is loaded at boot. Write operation at this address is possible only after system boot.

Table 42. Interrupt mode

AOI	6D	Interrupt mode
0	0	OR combination of interrupt events
0	1	6-direction movement recognition
1	0	AND combination of interrupt events
1	1	6-direction position recognition

Difference between AOI-6D = '01' and AOI-6D = '11'.

AOI-6D = '01' is movement recognition. An interrupt is generated when orientation moves from unknown zone to known zone. The interrupt signal stays for a duration ODR.

AOI-6D = '11' is direction recognition. An interrupt is generated when orientation is inside a known zone. The interrupt signal stays until orientation is inside the zone.

7.1.15 INT1_SRC_A (31h)

Table 43. INT1_SRC_A register

0(1)	IA	ZH	ZL	YH	YL	XH	XL
------	----	----	----	----	----	----	----

1. This bit must be set to '0' for correct working of the device.

Table 44. INT1_SRC_A description

IA	Interrupt active. Default value: 0 (0: no interrupt has been generated, 1: one or more interrupts have been generated)
ZH	Z high. Default value: 0 (0: no interrupt, 1: Z high event has occurred)
ZL	Z low. Default value: 0 (0: no interrupt, 1: Z low event has occurred)
YH	Y high. Default value: 0 (0: no interrupt, 1: Y high event has occurred)

Table 44. INT1_SRC_A description (continued)

YL	Y low. Default value: 0 (0: no interrupt, 1: Y low event has occurred)
XH	X high. Default value: 0 (0: no interrupt, 1: X high event has occurred)
XL	X low. Default value: 0 (0: no interrupt, 1: X low event has occurred)

Interrupt 1 source register. Read only register.

Reading at this address clears the INT1_SRC IA bit (and the interrupt signal on the INT 1 pin) and allows the refreshing of data in the INT1_SRC register if the latched option was chosen.

7.1.16 INT1_THS_A (32h)

Table 45. INT1_THS_A register

7.1.17

0 ⁽¹⁾	THS6	THS5	THS4	THS3	THS2	THS1	THS0
------------------	------	------	------	------	------	------	------

This bit must be set to '0' for correct working of the device.

Table 46.INT1_THS_A description

THS6 - THS0	Interrupt 1 threshold. Default value: 000 0000
-------------	--

INT1_DURATION_A (33h)**Table 47.INT1_DURATION_A register**

0 ⁽¹⁾	D6	D5	D4	D3	D2	D1	D0
------------------	----	----	----	----	----	----	----

This bit must be set to '0' for correct working of the device.

Table 48.INT1_DURATION_A description

D6 - D0	Duration value. Default value: 000 0000
---------	---

D6 - D0 bits set the minimum duration of the Interrupt 1 event to be recognized. Duration steps and maximum values depend on the ODR chosen.

7.1.18 **INT2_CFG_A (34h)****Table 49.INT2_CFG_A register**

AOI	6D	ZHIE	ZLIE	YHIE	YLIE	XHIE	XLIE
-----	----	------	------	------	------	------	------

Table 50.INT2_CFG_A description

AOI	AND/OR combination of interrupt events. Default value: 0 (see Table 51)
6D	6-direction detection function enabled. Default value: 0 (refer to Table 51)
ZHIE	Enable interrupt generation on Z high event. Default value: 0 (0: disable interrupt request, 1: enable interrupt request on measured accel. value higher than preset threshold)
ZLIE	Enable interrupt generation on Z low event. Default value: 0 (0: disable interrupt request, 1: enable interrupt request on measured accel. value lower than preset threshold)

YHIE	Enable interrupt generation on Y high event. Default value: 0 (0: disable interrupt request, 1: enable interrupt request on measured accel. value higher than preset threshold)
YLIE	Enable interrupt generation on Y low event. Default value: 0 (0: disable interrupt request, 1: enable interrupt request on measured accel. value lower than preset threshold)
XHIE	Enable interrupt generation on X high event. Default value: 0 (0: disable interrupt request, 1: enable interrupt request on measured accel. value higher than preset threshold)
XLIE	Enable interrupt generation on X low event. Default value: 0 (0: disable interrupt request, 1: enable interrupt request on measured accel. value lower than preset threshold)

Table 51. Interrupt mode

AOI	6D	Interrupt mode
0	0	OR combination of interrupt events
0	1	6-direction movement recognition
1	0	AND combination of interrupt events
1	1	6-direction position recognition

Difference between AOI-6D = '01' and AOI-6D = '11'.

AOI-6D = '01' is movement recognition. An interrupt is generated when orientation moves from unknown zone to known zone. The interrupt signal stays for a duration ODR.

AOI-6D = '11' is direction recognition. An interrupt is generated when orientation is inside a known zone. The interrupt signal stays until orientation is inside the zone.

7.1.19 INT2_SRC_A (35h)

Table 52. INT2_SRC_A register

0(1)	IA	ZH	ZL	YH	YL	XH	XL
------	----	----	----	----	----	----	----

1. This bit must be set to '0' for correct working of the device.

Table 53. INT2_SRC_A description

IA	Interrupt active. Default value: 0 (0: no interrupt has been generated, 1: one or more interrupts have been generated)
ZH	Z high. Default value: 0 (0: no interrupt, 1: Z high event has occurred)

ZL	Z low. Default value: 0 (0: no interrupt, 1: Z low event has occurred)
YH	Y high. Default value: 0 (0: no interrupt, 1: Y high event has occurred)
YL	Y low. Default value: 0 (0: no interrupt, 1: Y low event has occurred)
XH	X high. Default value: 0 (0: no interrupt, 1: X high event has occurred)
XL	X Low. Default value: 0 (0: no interrupt, 1: X low event has occurred)

Interrupt 2 source register. Read only register.

Reading at this address clears INT2_SRC IA bit (and the interrupt signal on the INT 2 pin) and allows the refreshing of data in the INT2_SRC register if the latched option was chosen.

7.1.20 INT2_THS_A (36h)

Table 54.INT2_THS_A register

7.1.21

0 ⁽¹⁾	THS6	THS5	THS4	THS3	THS2	THS1	THS0
------------------	------	------	------	------	------	------	------

This bit must be set to '0' for correct working of the device

Table 55.INT2_THS_A description

THS6 - THS0	Interrupt 1 threshold. Default value: 000 0000
-------------	--

INT2_DURATION_A (37h)

Table 56.INT2_DURATION_A register

0 ⁽¹⁾	D6	D5	D4	D3	D2	D1	D0
------------------	----	----	----	----	----	----	----

This bit must be set to '0' for correct working of the device

Table 57.INT2_DURATION_A description

D6-D0	Duration value. Default value: 000 0000
-------	---

D6 - D0 bits set the minimum duration of the Interrupt 2 event to be recognized. Duration time steps and maximum values depend on the ODR chosen.

7.1.22 CLICK_CFG_A (38h)

Table 58.CLICK_CFG_A register

--	--	ZD	ZS	YD	YS	XD	XS
----	----	----	----	----	----	----	----

Table 59. CLICK_CFG_A description

ZD	Enable interrupt double CLICK on Z axis. Default value: 0 (0: disable interrupt request, 1: enable interrupt request on measured accel. value higher than preset threshold)
ZS	Enable interrupt single CLICK on Z axis. Default value: 0 (0: disable interrupt request, 1: enable interrupt request on measured accel. value higher than preset threshold)
YD	Enable interrupt double CLICK on Y axis. Default value: 0 (0: disable interrupt request, 1: enable interrupt request on measured accel. value higher than preset threshold)
YS	Enable interrupt single CLICK on Y axis. Default value: 0 (0: disable interrupt request, 1: enable interrupt request on measured accel. value higher than preset threshold)

XD	Enable interrupt double CLICK on X axis. Default value: 0 (0: disable interrupt request, 1: enable interrupt request on measured accel. value higher than preset threshold)
XS	Enable interrupt single CLICK on X axis. Default value: 0 (0: disable interrupt request, 1: enable interrupt request on measured accel. value higher than preset threshold)

7.1.23 CLICK_SRC_A (39h)

Table 60.CLICK_SRC_A register

--	IA	DCLICK	SCLICK	Sign	Z	Y	X
----	----	--------	--------	------	---	---	---

Table 61.CLICK_SRC_A description

IA	Interrupt active. Default value: 0 (0: no interrupt has been generated, 1: one or more interrupts have been generated)
DCLICK	Double CLICK-CLICK enable. Default value: 0 (0:double CLICK-CLICK detection disable, 1: double CLICK-CLICK detection enable)
SCLICK	Single CLICK-CLICK enable. Default value: 0 (0:Single CLICK-CLICK detection disable, 1: single CLICK-CLICK detection enable)
Sign	CLICK-CLICK Sign. 0: positive detection, 1: negative detection

Table 61.CLICK_SRC_A description (continued)

Z	Z CLICK-CLICK detection. Default value: 0 (0: no interrupt, 1: Z high event has occurred)
Y	Y CLICK-CLICK detection. Default value: 0 (0: no interrupt, 1: Y high event has occurred)
X	X CLICK-CLICK detection. Default value: 0 (0: no interrupt, 1: X high event has occurred)

7.1.24 CLICK_THS_A (3Ah)

Table 62.CLICK_THS_A register

--	Ths6	Ths5	Ths4	Ths3	Ths2	Ths1	Ths0
----	------	------	------	------	------	------	------

Table 63. CLICK_SRC_A description

Ths6-Ths0	CLICK-CLICK threshold. Default value: 000 0000
-----------	--

1 LSB = full-scale / 128. THS6 through THS0 define the threshold which is used by the system to start the click detection procedure. The threshold value is expressed over 7 bits as an unsigned number.

7.1.25 TIME_LIMIT_A (3Bh)

Table 64. TIME_LIMIT_A register

--	TLI6	TLI5	TLI4	TLI3	TLI2	TLI1	TLI0
----	------	------	------	------	------	------	------

Table

65. TIME_LIMIT_A description

TLI7-TLI0	CLICK-CLICK time limit. Default value: 000 0000
-----------	---

1 LSB = 1/ODR. TLI7 through TLI0 define the maximum time interval that can elapse between the start of the click detection procedure (the acceleration on the selected channel exceeds the programmed threshold) and when the acceleration goes back below the threshold.

7.1.26 TIME_LATENCY_A (3Ch)

Table 66. TIME_LATENCY_A register

TLA7	TLA6	TLA5	TLA4	TLA3	TLA2	TLA1	TLA0
------	------	------	------	------	------	------	------

Table

67. TIME_LATENCY_A description

TLA7-TLA0	CLICK-CLICK time latency. Default value: 000 0000
-----------	---

1 LSB = 1/ODR. TLA7 through TLA0 define the time interval that starts after the first click detection where the click detection procedure is disabled, in cases where the device is configured for double click detection.

7.1.27 TIME_WINDOW_A (3Dh)

Table 68. TIME_WINDOW_A register

TW7	TW6	TW5	TW4	TW3	TW2	TW1	TW0
-----	-----	-----	-----	-----	-----	-----	-----

Table

69. TIME_WINDOW_A description

TW7-TW0	CLICK-CLICK time window
---------	-------------------------

1 LSB = 1/ODR. TW7 through TW0 define the maximum interval of time that can elapse after the end of the latency interval in which the click detection procedure can start, in cases where the device is configured for double click detection.

7.2 Magnetic field sensing register description

7.2.1 CRA_REG_M (00h)

Table 70.CRA_REG_M register

TEMP_EN	0(1)	0(1)	DO2	DO1	DO0	0(1)	0(1)
---------	------	------	-----	-----	-----	------	------

1. This bit must be set to '0' for correct working of the device

Table 71.CRA_REG_M description

TEMP_EN	Temperature sensor enable. 0: temperature sensor disabled (default), 1: temperature sensor enabled
DO2 to DO0	Data output rate bits. These bits set the rate at which data is written to all three data output registers (refer to Table 72). Default value: 100

Table 72.Data rate configurations

DO2	DO1	DO0	Minimum data output rate (Hz)
0	0	0	0.75
0	0	1	1.5
0	1	0	3.0
0	1	1	7.5
1	0	0	15
1	0	1	30

Table 72.Data rate configurations (continued)

DO2	DO1	DO0	Minimum data output rate (Hz)
1	1	0	75
1	1	1	220

7.2.2 CRB_REG_M (01h)

Table 73.CRA_REG register

GN2	GN1	GN0	0(1)	0(1)	0(1)	0(1)	0(1)
-----	-----	-----	------	------	------	------	------

1. This bit must be set to '0' for correct working of the device.

Table 74.CRA_REG description

GN1-0	Gain configuration bits. The gain configuration is common for all channels (refer to Table 75)
-------	---

Table 75. Gain setting

GN2	GN1	GN0	Sensor input field range [Gauss]	Gain X, Y, and Z [LSB/Gauss]	Gain Z [LSB/Gauss]	Output range
0	0	1	±1.3	1100	980	0xF800–0x07FF (-2048–2047)
0	1	0	±1.9	855	760	
0	1	1	±2.5	670	600	
1	0	0	±4.0	450	400	
1	0	1	±4.7	400	355	
1	1	0	±5.6	330	295	
1	1	1	±8.1	230	205	

7.2.3**MR_REG_M (02h)****Table 76. MR_REG**

0(1)	0(1)	0(1)	0(1)	0(1)	0(1)	MD1	MD0
------	------	------	------	------	------	-----	-----

1. This bit must be set to '0' for correct working of the device.

Table 77.MR_REG description

MD1-0	Mode select bits. These bits select the operation mode of this device (refer to Table 78)
-------	--

Table 78.Magnetic sensor operating mode

MD1	MD0	Mode
0	0	Continuous-conversion mode
0	1	Single-conversion mode
1	0	Sleep-mode. Device is placed in sleep-mode

1	1	Sleep-mode. Device is placed in sleep-mode
---	---	--

7.2.4 OUT_X_H_M (03), OUT_X_LH_M (04h)

X-axis magnetic field data. The value is expressed as 2's complement.

7.2.5 OUT_Z_H_M (05), OUT_Z_L_M (06h)

Z-axis magnetic field data. The value is expressed as 2's complement.

7.2.6 OUT_Y_H_M (07), OUT_Y_L_M (08h)

Y-axis magnetic field data. The value is expressed as 2's complement.

7.2.7 SR_REG_M (09h)**Table 79. SR register**

--	--	--	--	--	--	LOCK	DRDY
----	----	----	----	----	----	------	------

Table 80. SR register description

LOCK	Data output register lock. Once a new set of measurements is available, this bit is set when the first magnetic field data register has been read.
DRDY	Data ready bit. This bit is when a new set of measurements are available.

7.2.8 IR_REG_M (0Ah/0Bh/0Ch)**Table 81. IRA_REG_M**

0	1	0	0	1	0	0	0
---	---	---	---	---	---	---	---

Table 82. IRB_REG_M

0	0	1	1	0	1	0	0
---	---	---	---	---	---	---	---

Table 83. IRC_REG_M

0	0	1	1	0	0	1	1
---	---	---	---	---	---	---	---

7.2.9

TEMP_OUT_H_M (31h), TEMP_OUT_L_M (32h)

Table 84. TEMP_OUT_H_M register

TEMP11	TEMP10	TEMP9	TEMP8	TEMP7	TEMP6	TEMP5	TEMP4
--------	--------	-------	-------	-------	-------	-------	-------

Table 85. TEMP_OUT_L_M register

TEMP3	TEMP2	TEMP1	TEMP0	--	--	--	--
-------	-------	-------	-------	----	----	----	----

Table 86. TEMP_OUT resolution

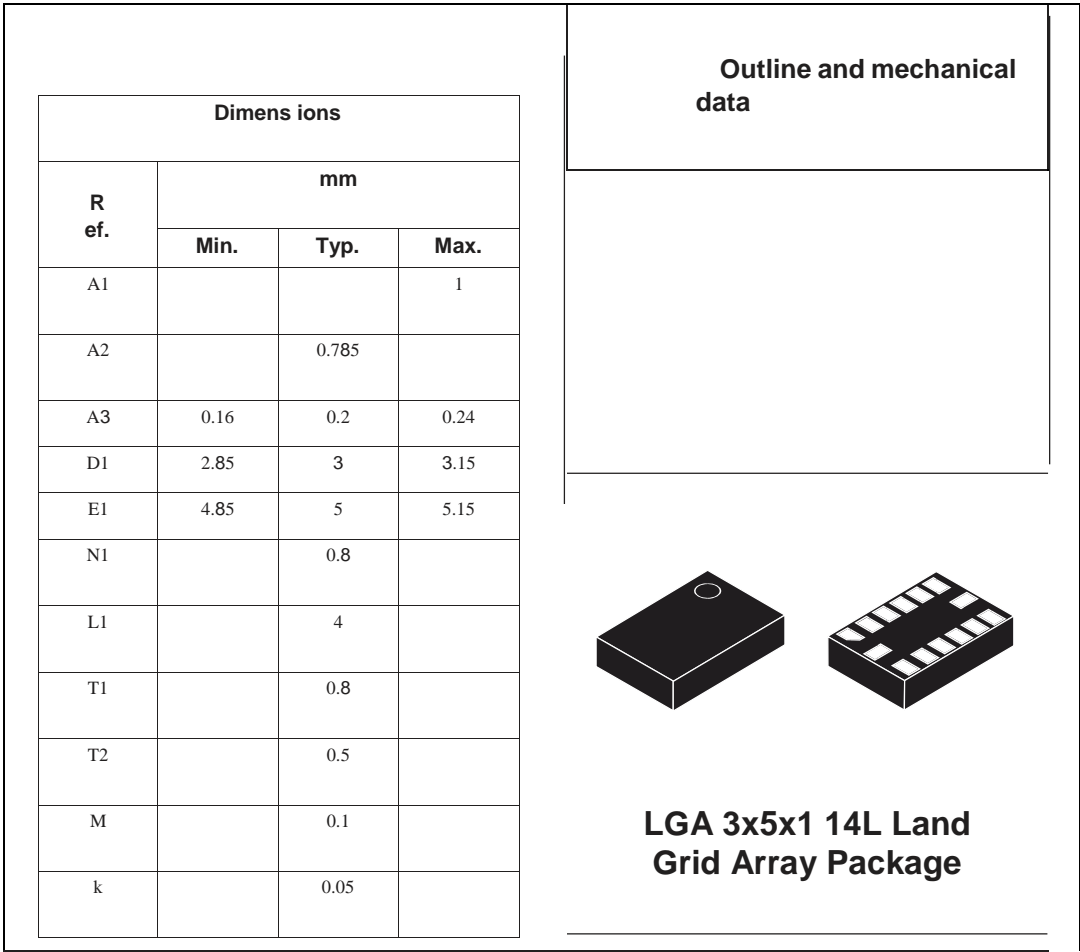
TEMP11-0	Temperature data (8LSB/deg - 12-bit resolution). The value is expressed as 2's complement.
----------	--

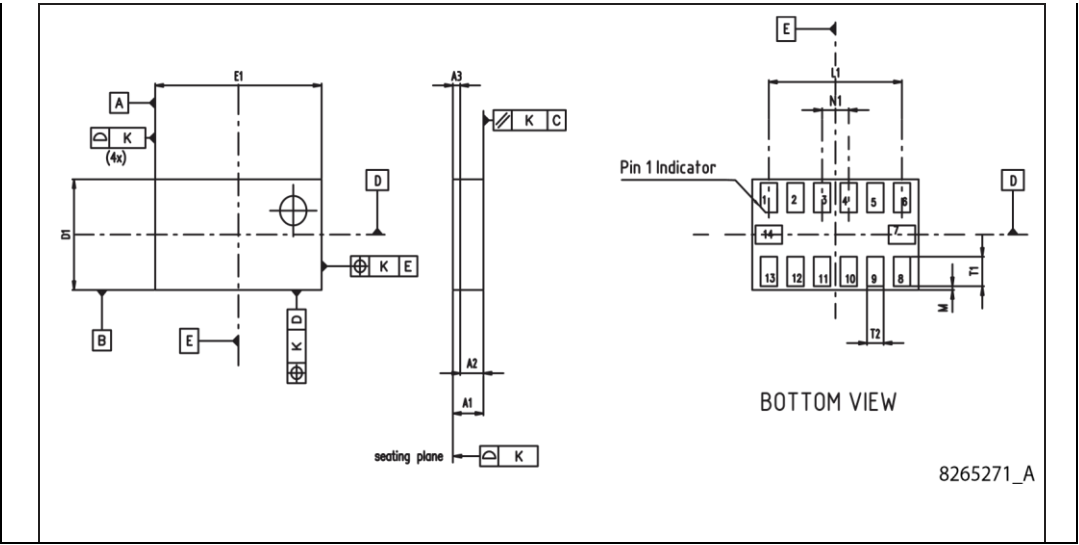
Package information

8 Package information

In order to meet environmental requirements, ST offers these devices in different grades of ECOPACK® packages, depending on their level of environmental compliance. ECOPACK specifications, grade definitions, and product status are available at: www.st.com. ECOPACK is an ST trademark.

Figure 5.LGA-14: mechanical data and package dimensions





Revision history

9 Revision history

Table 87.Document revision history

Date	Revision	Changes
21-Apr-2011	1	Initial release.

LSM303DLHC

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST's terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST'S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER'S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2011 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

MaxBotix Inc., products are engineered and assembled in the USA.

®

Page 75

MaxBotix Inc.

Copyright 2005 Patent 7,679,996- 2012 MaxBotix Incorporated

Web: www.maxbotix.com

PD11832b

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta
- Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

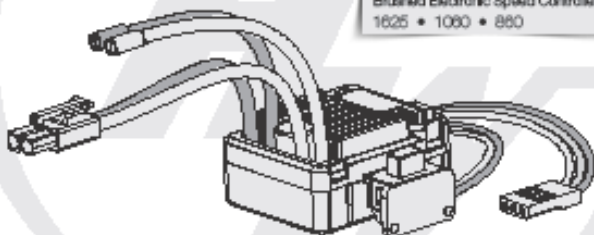
www.st.com



USER MANUAL

QUICRUN

Brushed Electronic Speed Controller
1625 • 1000 • 880



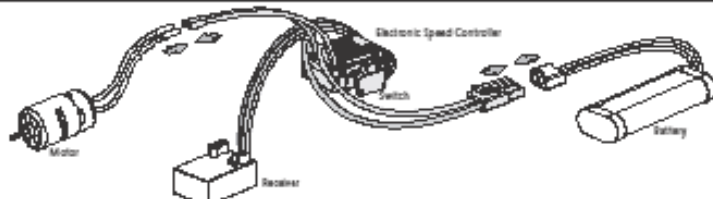
Congratulations and thanks for purchasing Hobbywing QUICRUN series electronic speed controller (ESC) for brushed motor. The power system for RC model can be very dangerous, so please read this manual carefully. Since we have no control over the installation, application, use or maintenance of this product, in no case shall we be liable for any damages, losses or costs.

01 Features

- Water-proof and dust-proof, suitable for all-weather condition races.
- Small size with built-in capacitor module.
- Three running modes: **Fwd/R**, **Fwd/Rev/R** and **Fwd/Rev**, fit for various vehicles.
- Note 1:** **Fwd** = Forward, **Rev** = Reverse, **R** = Reverse.
- Note 2:** QUICRUN-WP-1625-880-4ESD ESC only has the **Fwd / Rev / R** mode.
- Great current endurance capability.
- Great built-in ESC output capacity.
- Automatic throttle range calibration, easy to use.
- Easy to set the ESC parameters with jumpers.
- Multiple protections: Low voltage cut-off protection for battery / Over-heat protection / Throttle signal loss protection.

02 Begin to Use the New Brushed ESC

1 Connections



Turn off the ESC switch, wire the battery, motor, ESC, servo, receiver according to the following diagram. Check the wiring to ensure all connections are correct before getting into the next step.

1) Once the power is wrongly connected (that means the battery polarity is mistakenly reversed), irreparable damage may occur to the ESC and batteries. Therefore, please pay close attention to the battery polarity.

2) Please swap the two wire connections if the motor rotate in the opposite direction.

Specifications

Model	Quicrun-WP-1625-880-4ESD	Quicrun-WP-1616-BRUSHED	Quicrun-WP-880-DUAL-BRUSHED
Fwd. Cont. / Peak Current	25A/100A	60A/260A	60A/260A
Rev. Cont. / Peak Current	25A/100A	20A/180A	20A/180A
Voltage Range	2-25 Lipo or 5-6 NiMH		
Cars Applicable	1/18 & 1/16 Touring Car, Buggy, Monster, Truggy	1/10 Touring Car, Buggy, Short Course Truck, Monster, Truggy, Rock Crawler and Tank	1/8 Touring Car, Buggy, Short Course Truck, Monster, Truggy, Rock Crawler and Tank
Motor Limit *	25 Lipo or 6 NiMH 280, 270 or 260 Size Motor: RPM<20000 @7.2V 25 Lipo or 9 NiMH 280, 270 or 260 Size Motor: RPM<20000 @7.2V 4S Lipo or 12 NiMH Not Available	S40 or S20 Size Motor: >=12T or RPM<20000 @7.2V S40 or S20 Size Motor: >=18T or RPM<20000 @7.2V	S40, S20, 775 Size Motor: >=12T or RPM<20000 @7.2V S40, S20, 775 Size Motor: >=18T or RPM<20000 @7.2V S40, S20, 775 Size Motor: >=24T or RPM<20000 @7.2V
Resistance	Red 0.003Ω, Rev 0.003Ω	Red 0.001Ω, Rev 0.002Ω	Red 0.001Ω, Rev 0.002Ω
ESC Output	1A / 5V (Linear Mode)	2A / 5V (Linear Mode)	2A / 5V (Switch Mode)
PWM Frequency	10KHz		
Dimension / Weight	26x26x16mm / 23.5g	26.5x26.5x16mm / 29 g	26x26x26mm / 73g
Cooling Fan	Without cooling fan		
Running Modes	Forward / Reverse / Brake	Forward / Reverse / Brake, Forward / Brake, Forward / Reverse	Forward / Reverse / Brake, Forward / Brake, Forward / Reverse, Stall

*Note: WP-880-DUAL-BRUSHED has two outputs to drive 2 motors. When driving 2 motors simultaneously, the Torque of the motor need to be increased.

2 Set the Throttle Range

Turn on the transmitter, and set parameters (of the throttle channel) like "DR", "GRA", "ATL" to 100% (if there is no LCD display on the transmitter, please adjust the corresponding knob to its limit). Set the throttle trim to 0 (if there is no display, then adjust the knob to the neutral position). For FUTABA™ and similar transmitters, set the throttle direction to "REV", while the throttle direction of others to "FWD". Please disable the built-in ABS brake function in your transmitter.

Besides, we strongly recommend users to enable the "Fail Safe (FS)" function of the transmitter, set the "FS" of the throttle channel to the Shutdown mode or set the protection value to the neutral position, so the car can be stopped if the receiver fails to get the radio signals from the transmitter.

Calibrate the throttle range: Turn on the ESC switch, set the throttle stick to the neutral point and then wait 2 seconds for the completion of throttle range self-calibration. Bleep sound emits if the self-calibration is successfully passed, then the ESC is ready to run.

The Meaning of Bleep Sound	LCD Status (in Running)	Throttle Stick Position
<ul style="list-style-type: none"> • 1 short Bleep: The battery is NiMH • 2 short Bleep: The battery is 2S Lipo • 3 short Bleep: The battery is 3S Lipo • 4 short Bleep: The battery is 4S Lipo • 1 long Bleep: Self-test and throttle range calibration is OK, the ESC is ready to run. 	<ul style="list-style-type: none"> • When the throttle stick is in neutral range, red LED is off • Partial throttle forward, partial brake or partial reverse, red LED blinks • Full throttle forward, maximum brake or full throttle reverse, red LED is solid on 	<p>Turn on the switch</p> <p>Neutral point</p>

03 Set the ESC Parameters

How to Set Parameters:

1. QUICRUN-WP-1625 / 1000-4ESD ESC uses the jumper caps to set running mode & battery type. (Note: The "running mode" is not programmable for the QUICRUN-WP-1625-880-4ESD ESC.)

Way to set: We suggest users use the **hexcon** to set parameters by plugging / unplugging the jumper cap (as shown in the picture below). For example, if want set the battery type to the "Lipo" mode, you only need to plug the jumper cap into left two pins of the battery pin header.

2. QUICRUN-WP-880-DUAL-BRUSHED ESC uses the dial switch to set running mode & battery type.

Way to set: recommended users use the **hexcon** to set parameters by flipping the DIP switch (for detailed explanation, please refer to the following picture). If want to set the battery type to the "Lipo" mode, you only need to flip the battery dip switch to the left position.

Programmable Items

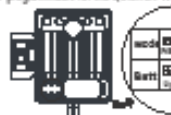
1. **Running Mode:** 3 Options (Fwd / Rr / Rev, Fwd / Rr, Fwd / Rev). The "Fwd / Rr / Rev" is the default option. **Fwd/Forward, Rr/Reverse, Rev/Reverse**. "Fwd / Rr / Rev" mode indicates the vehicle can go forward, backward and brake. This mode uses "Double-click" method to make the vehicle reverse. When moving the throttle stick from the neutral zone to backward zone for the 1st time, the ESC begins to brake the motor and the motor slows down but still running, so the backward action is NOT performed immediately. When the throttle stick is moved to the backward zone again, if the motor speed slows down to zero (i.e. stopped), the backward action will happen. This "Double-click" method prevents mistakenly reversing action when the brake function is frequently used in steering. Therefore, this mode is often used in daily practice. For the "Fwd / Rr" mode, the vehicle can go forward and brake, but no reversing, so this mode is often used in competitions. And the "Fwd / Rev" mode uses "Single-click" method to make the vehicle reverse, when moving the throttle stick from neutral zone to backward zone, the vehicle reverses immediately, so this mode is usually used for rock crawler. (Note: WP-1625-880-4ESD has no optional running mode except the default "Fwd / Rr / Rev" mode.)

"Stall" mode: this mode uses some brand-new algorithm that is specially designed for RC boats. (Only the WP-880-DUAL-BRUSHED ESC has this option in its programmable items.)

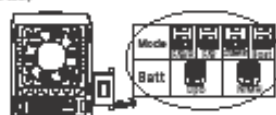
2. **Battery Type:** 2 Options (Lipo, NiMH), the "Lipo" is the default option.



WP-1625-BRUSHED



WP-1000-DUAL-BRUSHED



WP-880-DUAL-BRUSHED

04 Protection Features

1. **Low-Voltage Cut-off Protection:** If the voltage of battery pack is lower than the threshold for 2 seconds, the ESC will enter the protection mode, so the motor speed will be slowed when voltage is lower than the 1st trigger point (if stopped when voltage is lower than the 2nd trigger point). When the car stops, the red LED blinks to indicate the low voltage cut-off protection has been activated.

2S Lipo	3S Lipo	4S Lipo	5-6 NiMH
When the voltage is below 6.0V, the output power will be halved. When the voltage is lower than 5.0V, the output will be cut off and won't be resumed again.	When the voltage is below 6.70V, the output power will be halved. When the voltage is lower than 5.9V, the output will be cut off and won't be resumed again.	When the voltage is below 12.0V, the output power will be halved. When the voltage is lower than 11.0V, the output will be cut off and won't be resumed again.	When the voltage is below 8.5V, the output power will be halved. When the voltage is lower than 8.0V, the output will be cut off and won't be resumed again.

Note: when using the WP-880-DUAL-BRUSHED ESC to the "Stall" mode, the motor will stop running when the LVC protection is activated. Please move the throttle stick to neutral position, after that the motor can be started up again but the output power will be limited.

2. **Over-heat Protection:** When the internal temperature of the ESC is higher than 100 Celsius degrees, this protection will be activated and the output power will be reduced 90% cut off. The red LED blinks when the vehicle stops, and the ESC will not resume output power until its temperature is below 40 Celsius degrees.

3. **Throttle signal loss protection:** The ESC will cut off the output power if the throttle signal has been lost for 0.1 second. The "Fail Safe" function of the radio system is strongly recommended to be activated.


UNISONIC TECHNOLOGIES CO., LTD
U18
LINEAR INTEGRATED CIRCUIT

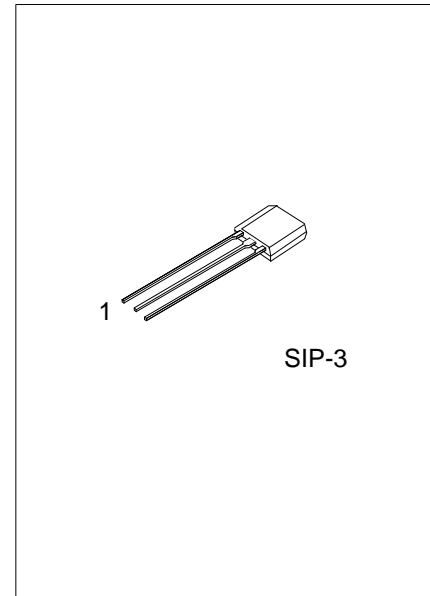
BIPOLAR LATCH TYPE HALL-EFFECT FOR HIGH-TEMPERATURE OPERATION

DESCRIPTION

U18 is a semiconductor integrated circuit utilizing the Hall effect. It has been so designed as to operate in the alternating magnetic field especially at low supply voltage and operation over extended temperature ranges to +125°C. This Hall IC is suitable for application to various kinds of sensors, contact less switches, and the like.

FEATURES

- * Wide supply voltage range of 2.5V to 20V
- * Wide temperature operation range of -20°C ~ +125



*Pb-free plating product number: U18L



MaxBotix Inc.

Copyright 2005 Patent 7,679,996- 2012 MaxBotix Incorporated

MaxBotix Inc., products are engineered and assembled in the USA.

Page 79

Web: www.maxbotix.com

PD11832b

Copyright © 2005 Unisonic Technologies Co., Ltd

QW-R118-007,A

- * Alternating magnetic field operation
- * TTL and MOS IC are directly drivable by the output
- * The life is semipermanent because it employs contact less parts
- * SIP-3 package www.DataSheet4U.com

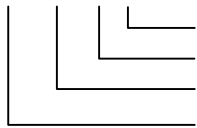
APPLICATION

- * Speed sensor
- * Position sensor
- * Rotation sensor
- * Contact-less sensor
- * Motor control
- * Built-in protection diode

ORDERING INFORMATION

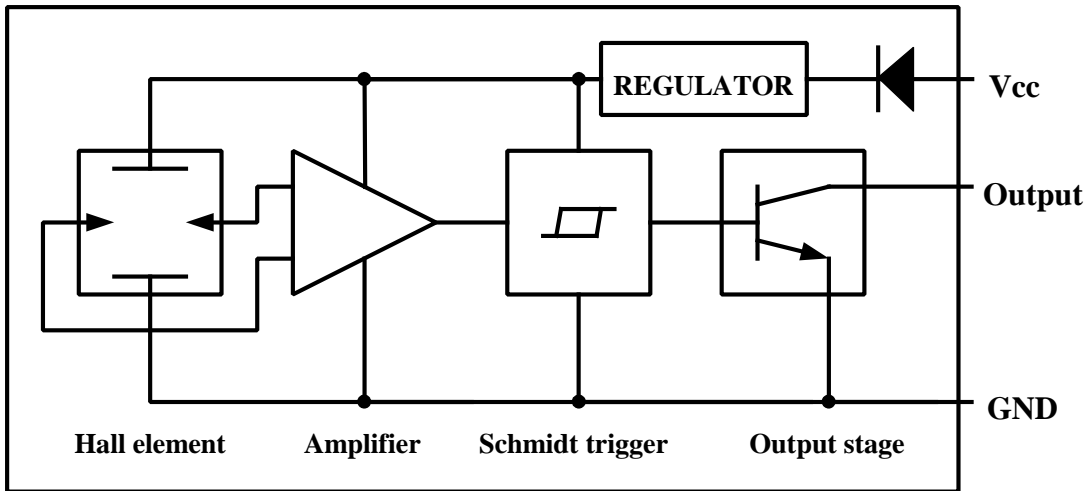
Order Number		Package	Pin Assignment			Packing
Normal	Lead Free Plating		1	2	3	
U18-G03-D-K	U18L-G03-D-K	SIP-3	I	G	O	Bulk

Note: Pin Assignment: I:V_{CC} O:V_{OUT} G:GND

 <p>U18L-G03-D-K (1)Packing Type (2)Pin Assignment (3)Package Type (4)Lead Plating</p>	<p>(1) K: Bulk (2) refer to Pin Assignment (3) G03: SIP-3 (4) L: Lead Free Plating, Blank: Pb/Sn</p>
--	---

MARKING INFORMATION



BLOCK DIAGRAM**ABSOLUTE MAXIMUM RATINGS**

PARAMETER	SYMBOL	RATING	UNIT
Supply Voltage	V_{CC}	2.5V ~ 20V	V
Supply Current	I_{CC}	10	mA
Circuit Current	I_{OUT}	20	mA
Power Dissipation	P_D	400	mW
Operating Temperature	T_{OPR}	-20~+125	°C
Storage Temperature	T_{STG}	-40~+150	°C

Note 1. Absolute maximum ratings are those values beyond which the device could be permanently damaged.

Absolute maximum ratings are stress ratings only and functional device operation is not implied.

2. The device is guaranteed to meet performance specification within 0°C~+70°C operating temperature range and assured by design from -20°C~+125°C.

ELECTRICAL CHARACTERISTICS ($T_a=25^\circ\text{C}$)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNIT
Low-Level Output Voltage	V_{OL}	$V_{CC}=16V, I_{OUT}=12mA, B=30mT$			0.7	V
		$V_{CC}=3.6V, I_{OUT}=12mA, B=30mT$			0.7	V
Output Leakage Current	$I_{O(LEAK)}$	$V_{CC}=16V, B=-30mT$		1	10	μA
Output Short Circuit Current	$-I_{OS}$	$V_{CC}=16V, V_{OUT}=0V, B=-30mT$		0.8		mA
Supply Current	I_{CC}	$V_{CC}=16V$			6	mA
		$V_{CC}=3.6V$			5.5	mA

MAGNETIC CHARACTERISTICS

Operate Point	B_{OP}	At $T_a = +25^\circ\text{C}$			5	mT
Release Point	B_{RP}	At $T_a = +25^\circ\text{C}$			-5	mT
Hysteresis	B_{HYS}	At $T_a = +25^\circ\text{C}$			5.5	mT

Note 1. B_{OP} = operate point (output turns ON); B_{RP} = release point (output turns OFF); B_{HYS} = hysteresis ($B_{OP} - B_{RP}$). As used here, negative flux densities are defined as less than zero (algebraic convention).

Typical values are at $T_a = +25^{\circ}\text{C}$ and $V_{CC} = 12\text{V}$.

2.1mT=10 gauss

PACKAGE INFORMATION

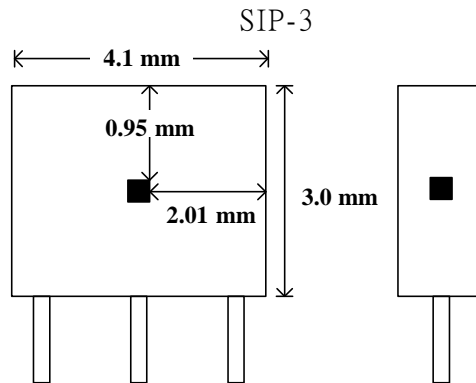


Fig. 1 SENSOR LOCATIONS

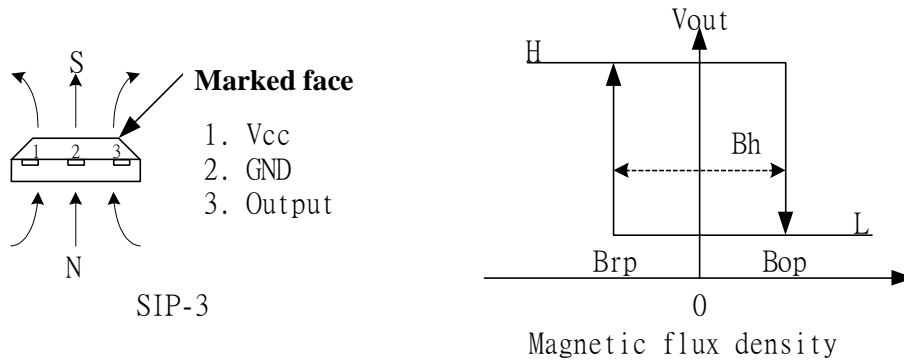
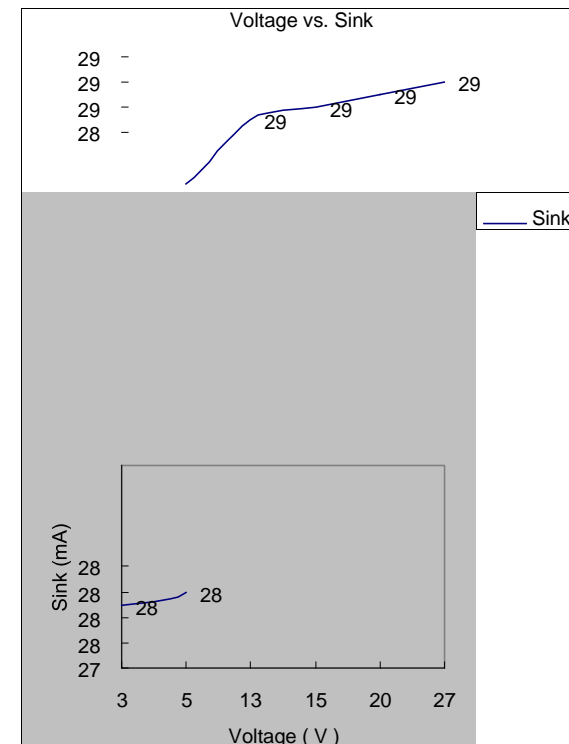
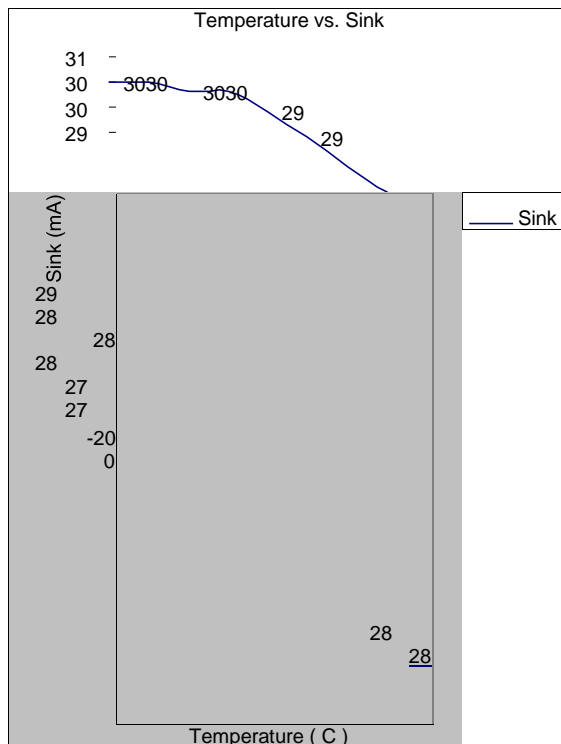
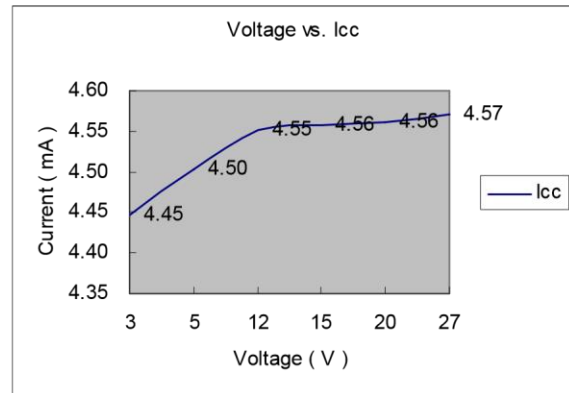
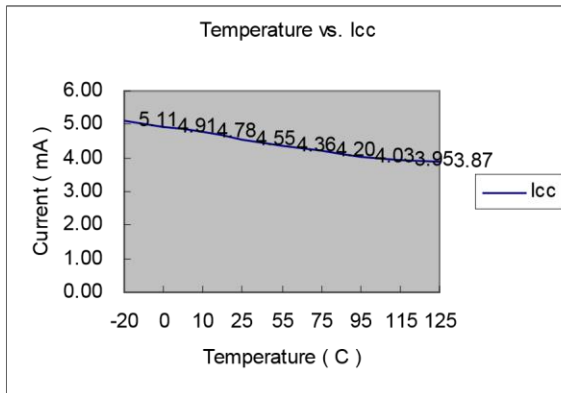
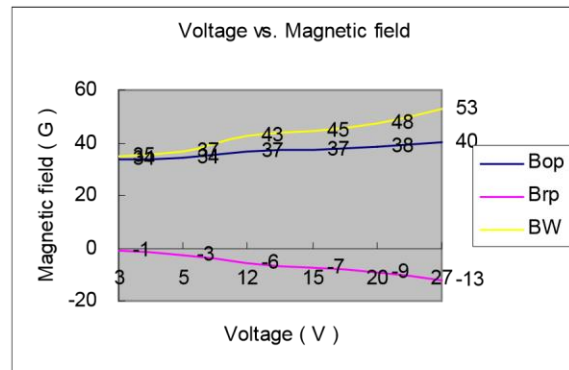
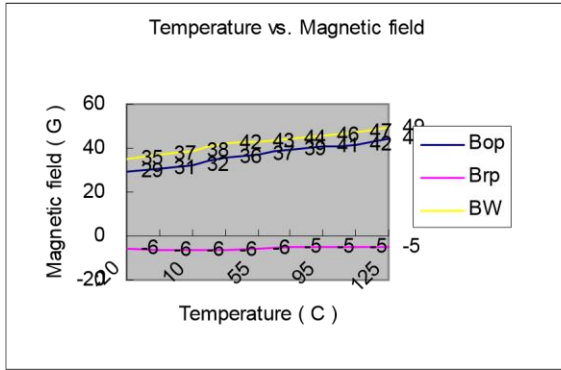
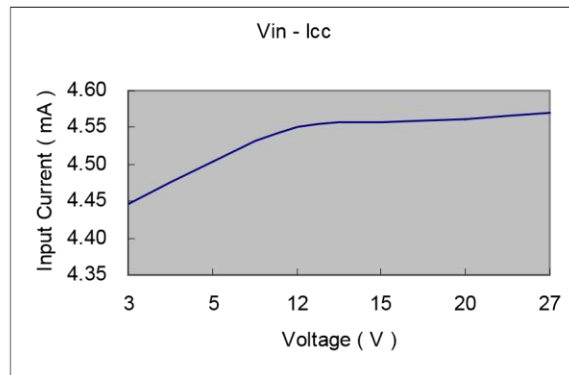
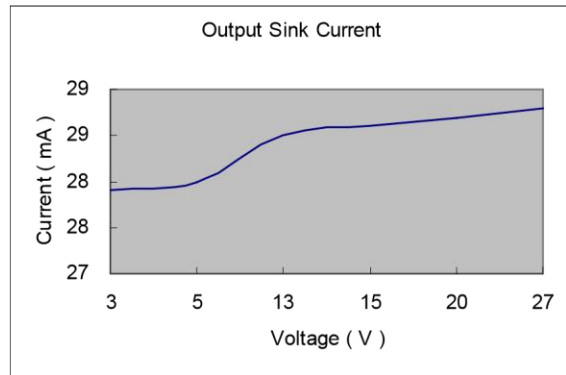
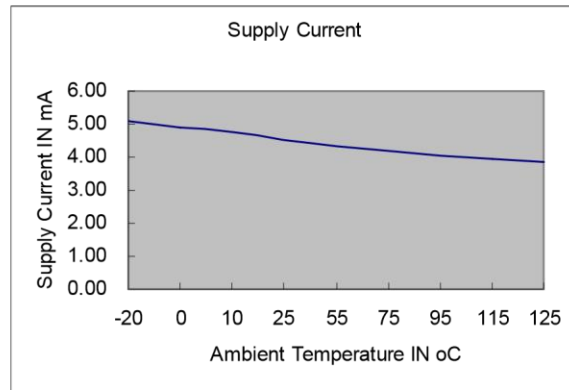
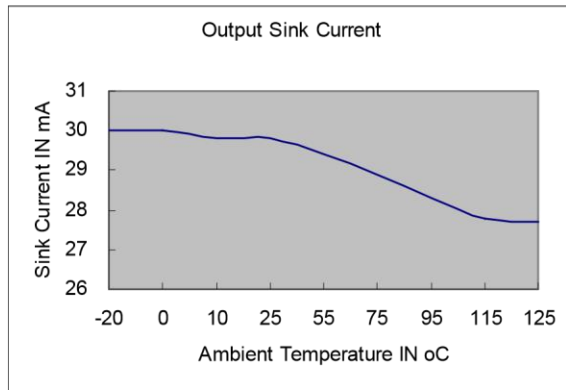
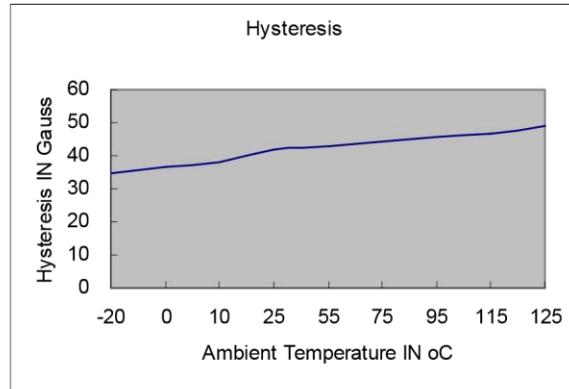
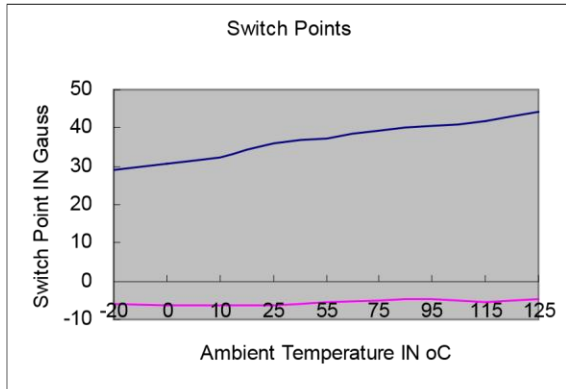


Fig. 2 APPLYING DIRECTION OF MAGNETIC FLUX

TYPICAL CHARACTERISTICS



TYPICAL CHARACTERISTICS(Cont.)



UTC assumes no responsibility for equipment failures that result from using products at values that exceed, even momentarily, rated values (such as maximum ratings, operating condition ranges, or other parameters) listed in products specifications of any and all UTC products described or contained herein. UTC products are not designed for use in life support appliances, devices or systems where malfunction of these products can be reasonably expected to result in personal injury. Reproduction in whole or in part is prohibited without the prior written consent of the copyright owner. The information presented in this document does not form part of any quotation or contract, is believed to be accurate and reliable and may be changed without notice.

LV-MaxSonar®-EZ™ Series

High Performance Sonar Range Finder

MB1000, MB1010, MB1020, MB1030, MB1040

With 2.5V - 5.5V power the LV-MaxSonar-EZ provides very short to long-range detection and ranging in a very small package. The LV-MaxSonar-EZ detects objects from 0-inches to 254-inches (6.45-meters) and provides sonar range information from 6-inches out to 254-inches with 1-inch resolution. Objects from 0-inches to 6-inches typically range as 6-inches¹. The interface output formats included are pulse width output, analog voltage output, and RS232 serial output. Factory calibration and testing is completed with a flat object. ¹See Close Range Operation



Features

- Learns ringdown pattern when
- Sensor reports the range reading

processor

- Continuously variable gain for sensor outputs control and side lobe suppression
- Object detection to zero range objects or internally
- 2.5V to 5.5V supply with 2mA

and Uses

typical current draw

- Readings can occur up to every 50mS, (20-Hz rate)
- Free run operation can continually

commanded to start ranging

directly and frees up user

- Designed for protected indoor environments
- Sensor operates at 42KHz
- High output square wave sensor drive

- Choose one of three
- Triggered externally

Applications

- (double Vcc) □ UAV blimps, micro planes and some helicopters

Benefits

- Very low cost ultrasonic rangefinder

- Bin level measurement
- Proximity

zone detection measure and output range

- Reliable and stable range data

- People

detection

information

- Quality beam characteristics

- Robot ranging sensor

®

MaxBotix Inc., products are engineered and assembled in the USA.

Page 85

MaxBotix Inc.

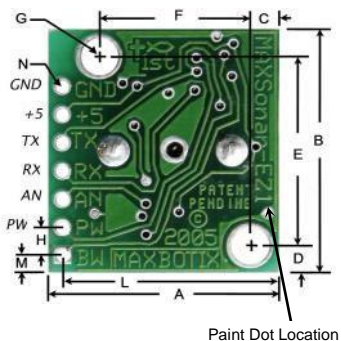
Copyright 2005 Patent 7,679,996- 2012 MaxBotix Incorporated

Web: www.maxbotix.com

PD11832b

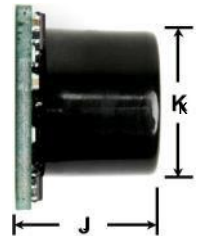
- Triggered operation provides the □ Mounting holes provided on the range reading as desired circuit board □ Autonomous navigation
- Interfaces are active simultaneously □ Very low power ranger, excellent for □ Multi-sensor arrays
- Serial, 0 to Vcc, 9600 Baud, 81N □ multiple sensor or battery-based □ Distance measuring
- Analog, (Vcc/512) / inch systems □ Long range object detection
- Pulse width, (147uS/inch) □ Fast measurement cycles □ Wide beam sensitivity

LV-MaxSonar-EZ Mechanical Dimensions



A	0.785"	19.9 mm	H	0.100"	2.54 mm
B	0.870"	22.1 mm	J	0.610"	15.5 mm
C	0.100"	2.54 mm	K	0.645"	16.4 mm
D	0.100"	2.54 mm	L	0.735"	18.7 mm
E	0.670"	17.0 mm	M	0.065"	1.7 mm
F	0.510"	12.6 mm	N	0.038" dia.	1.0 mm dia.
G	0.124" dia.	3.1 mm dia.	weight, 4.3 grams		

Part Number	MB1000	MB1010	MB1020	MB1030	MB1040
Paint Dot Color	Black	Brown	Red	Orange	Yellow



Close Range Operation

Applications requiring 100% reading-to-reading reliability should not use MaxSonar sensors at a distance closer than 6 inches. Although most users find MaxSonar sensors to work reliably from 0 to 6 inches for detecting objects in many applications, MaxBotix® Inc. does not guarantee operational reliability for objects closer than the minimum reported distance. Because of ultrasonic physics, these sensors are unable to achieve 100% reliability at close distances.

Warning: Personal Safety Applications

We do not recommend or endorse this product be used as a component in any personal safety applications. This product is not designed, intended or authorized for such use. These sensors and controls do not include the self-checking redundant circuitry needed for such use. Such unauthorized use may create a failure of the MaxBotix® Inc. product which may result in personal injury or death. MaxBotix® Inc. will not be held liable for unauthorized use of this component.

About Ultrasonic Sensors

Our ultrasonic sensors are in air, non-contact object detection and ranging sensors that detect objects within an area. These sensors are not affected by the color or other visual characteristics of the detected object. Ultrasonic sensors use high frequency sound to detect and localize objects

in a variety of environments. Ultrasonic sensors measure the time of flight for sound that has been transmitted to and reflected back from nearby objects. Based upon the time of flight, the sensor then outputs a range reading.

Pin Out Description

Pin 1-BW-*Leave open or hold low for serial output on the TX output. When BW pin is held high the TX output sends a pulse (instead of serial data), suitable for low noise chaining.

Pin 2-PW- This pin outputs a pulse width representation of range. The distance can be calculated using the scale factor of 147uS per inch.

Pin 3-AN- Outputs analog voltage with a scaling factor of (Vcc/512) per inch. A supply of 5V yields ~9.8mV/in. and 3.3V yields ~6.4mV/in. The output is buffered and corresponds to the most recent range data.

Pin 4-RX- This pin is internally pulled high. The LV-MaxSonar-EZ will continually measure range and output if RX data is left unconnected or held high. If held low the sensor will stop ranging. Bring high for 20uS or more to command a range reading.

Pin 5-TX- When the *BW is open or held low, the TX output delivers asynchronous serial with an RS232 format, except voltages are 0-Vcc. The output is an ASCII capital "R", followed by three ASCII character digits representing the range in inches up to a maximum of 255, followed by a carriage return (ASCII 13). The baud rate is 9600, 8 bits, no parity, with one stop bit. Although the voltage of 0-Vcc is outside the RS232 standard, most RS232 devices have sufficient margin to read 0-Vcc serial data. If standard voltage level RS232 is desired, invert, and connect an RS232 converter such as a MAX232. When BW pin is held high the TX output sends a single pulse, suitable for low noise chaining. (no serial data)

Pin 6-+5V- Vcc – Operates on 2.5V - 5.5V. Recommended current capability of 3mA for 5V, and 2mA for 3V. **Pin 7-GND**- Return for the DC power supply. GND (& Vcc) must be ripple and noise free for best operation.

Range "0" Location

The LV-MaxSonar-EZ reports the range to distant targets starting from the front of the sensor as shown in the diagram below.



Range Zero

The range is measured from the front of the transducer.

In general, the LV-MaxSonar-EZ will report the range to the leading edge of the closest detectable object. Target detection has been characterized in the sensor beam patterns.

Sensor Minimum Distance

The sensor minimum reported distance is 6-inches (15.2 cm). However, the LV-MaxSonar-EZ will range and report targets to the front sensor face. Large targets closer than 6-inches will typically range as 6-inches.

Sensor Operation from 6-inches to 20-inches

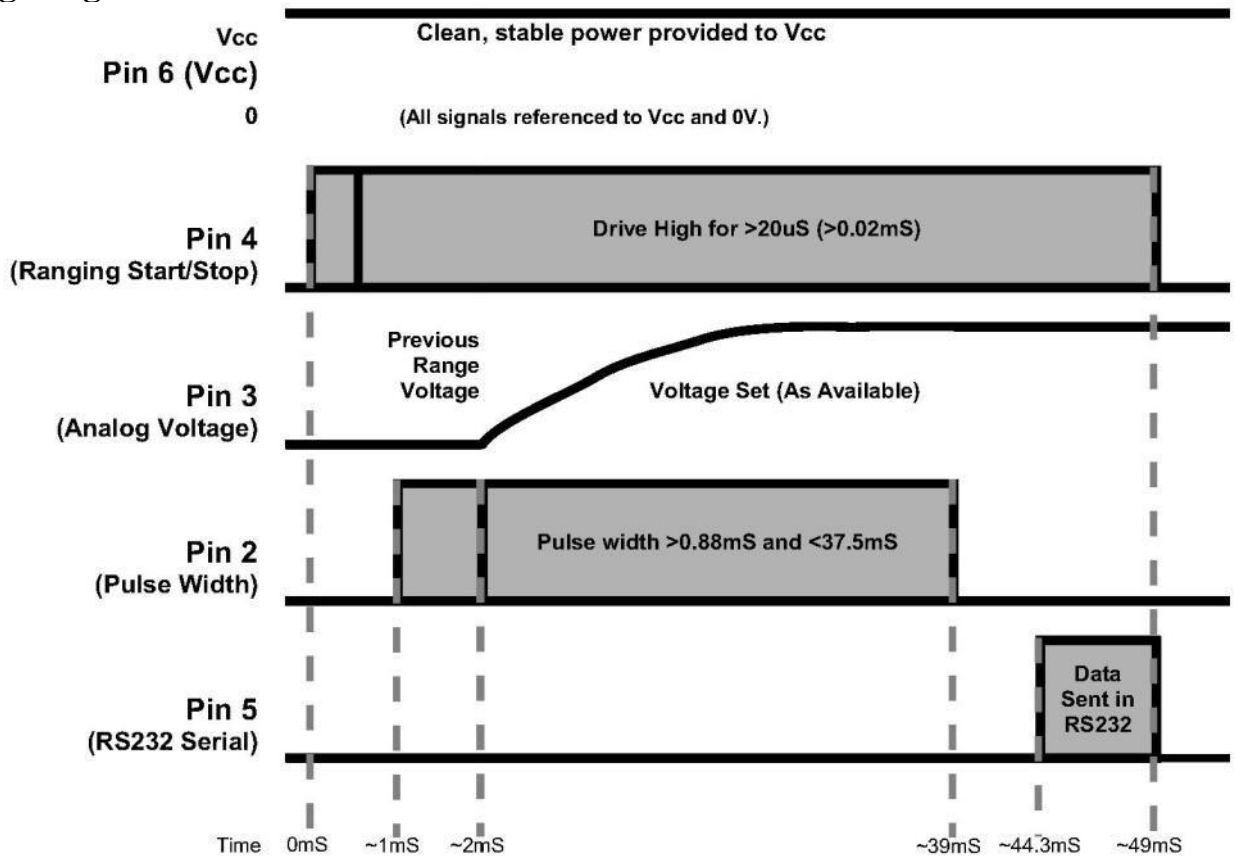
Because of acoustic phase effects in the near field, objects between 6-inches and 20-inches may experience acoustic phase cancellation of the returning waveform resulting in inaccuracies of up to 2-inches. These effects become less prevalent as the target distance increases, and has not been observed past 20-inches.

General Power-Up Instruction

Each time the LV-MaxSonar-EZ is powered up, it will calibrate during its first read cycle. The sensor uses this stored information to range a close object. It is important that objects not be close to the sensor during this calibration cycle. The best sensitivity is obtained when the detection area is clear for fourteen inches, but good results are common when clear for at least seven inches. If an object is too close during the calibration cycle, the sensor may ignore objects at that distance.

The LV-MaxSonar-EZ does not use the calibration data to temperature compensate for range, but instead to compensate for the sensor ringdown pattern. If the temperature, humidity, or applied voltage changes during operation, the sensor may require recalibration to reacquire the ringdown pattern. Unless recalibrated, if the temperature increases, the sensor is more likely to have false close readings. If the temperature decreases, the sensor is more likely to have reduced up close sensitivity. To recalibrate the LV-MaxSonar-EZ, cycle power, then command a read cycle.

Timing Diagram



Timing Description

250mS after power-up, the LV-MaxSonar-EZ is ready to accept the RX command. If the RX pin is left open or held high, the sensor will first run a calibration cycle (49mS), and then it will take a range reading (49mS). After the power up delay, the first reading will take an additional ~100mS. Subsequent readings will take 49mS. The LV-MaxSonar-EZ checks the RX pin at the end of every cycle. Range data can be acquired once every 49mS.

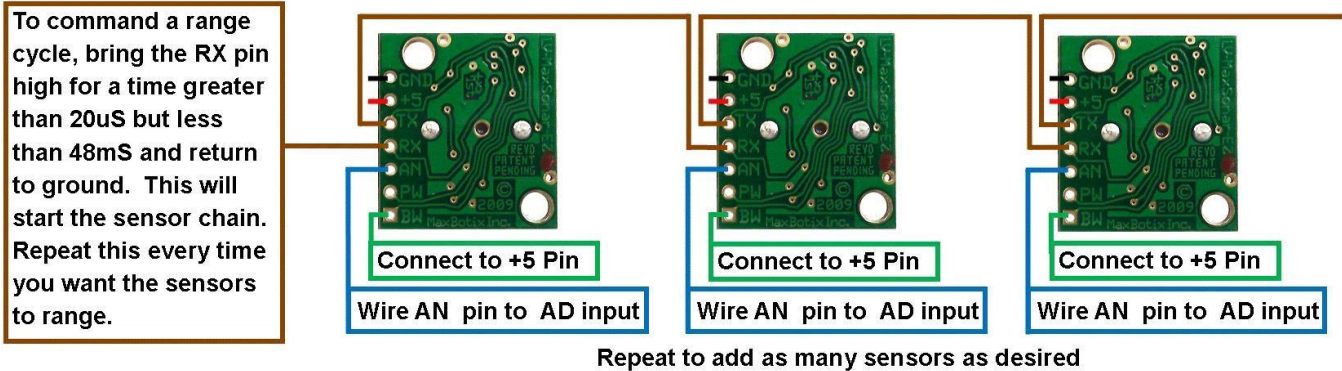
Each 49mS period starts by the RX being high or open, after which the LV-MaxSonar-EZ sends the transmit burst, after which the pulse width pin (PW) is set high. When a target is detected the PW pin is pulled low. The PW pin is high for up to 37.5mS if no target is detected. The remainder of the 49mS time (less 4.7mS) is spent adjusting the analog voltage to the correct level. When a long distance is measured immediately after a short distance reading, the analog voltage may not reach the exact level within one read cycle. During the last 4.7mS, the serial data is sent.

The LV-MaxSonar-EZ timing is factory calibrated to one percent at five volts, and in use is better than two percent. In addition, operation at 3.3V typically causes the objects range, to be reported, one to two percent further than actual.

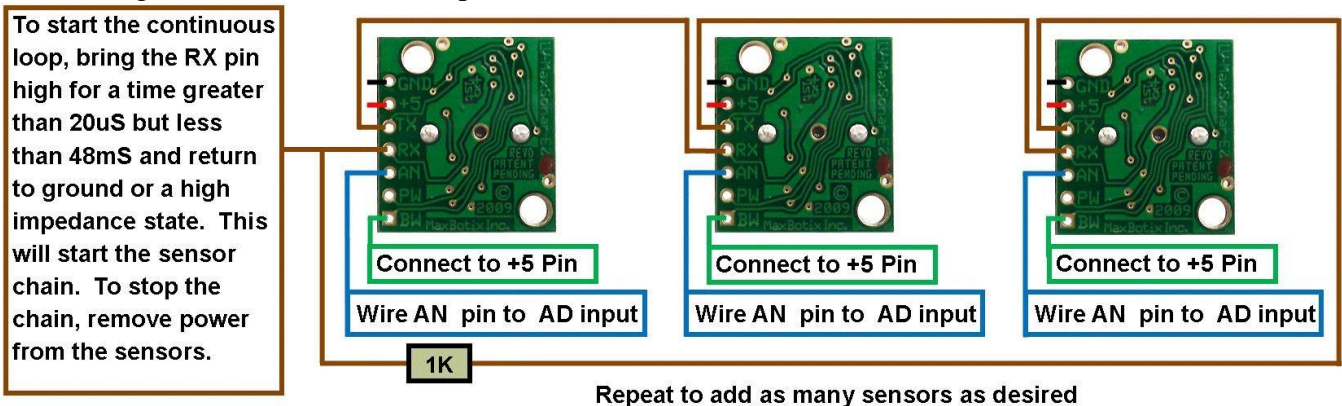
Using Multiple Sensors in a single system

When using multiple ultrasonic sensors in a single system, there can be interference (cross-talk) from the other sensors. MaxBotix Inc., has engineered and supplied a solution to this problem for the LV-MaxSonar-EZ sensors. The solution is referred to as chaining. We have 3 methods of chaining that work well to avoid the issue of cross-talk.

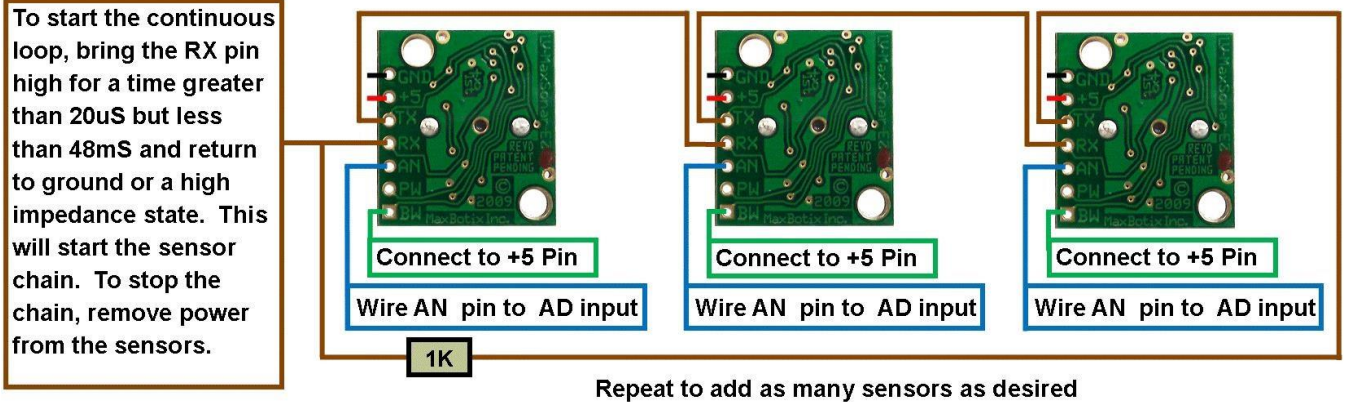
The first method is AN Output Commanded Loop. The first sensor will range, then trigger the next sensor to range and so on for all the sensor in the array. Once the last sensor has ranged, the array stops until the first sensor is triggered to range again. Below is a diagram on how to set this up.



The next method is AN Output Constantly Looping. The first sensor will range, then trigger the next sensor to range and so on for all the sensor in the array. Once the last sensor has ranged, it will trigger the first sensor in the array to range again and will continue this loop indefinitely. Below is a diagram on how to set this up.

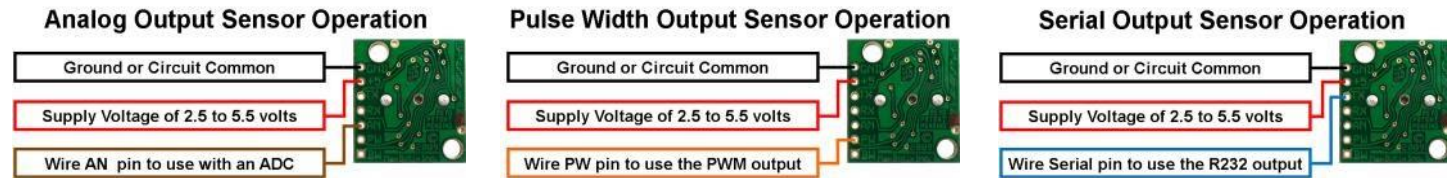


The final method is AN Output Simultaneous Operation. This method does not work in all applications and is sensitive to how the other sensors in the array are positioned in comparison to each other. Testing is recommend to verify this method will work for your application. All the sensors RX pins are conned together and triggered at the same time



Independent Sensor Operation

The LV-MaxSonar-EZ sensors have the capability to operate independently when the user desires. When using the LV-MaxSonar-EZ sensors in single or independent sensor operation, it is easiest to allow the sensor to free-run. Free-run is the default mode of operation for all of the MaxBotix Inc., sensors. The LV-MaxSonar-EZ sensors have three separate outputs that update the range data simultaneously: Analog Voltage, Pulse Width, and RS232 Serial. Below are diagrams on how to connect the sensor for each of the three outputs when operating in a single or independent sensor operating environment.



Selecting an LV-MaxSonar-EZ

Different applications require different sensors. The LV-MaxSonar-EZ product line offers varied sensitivity to allow you to select the best sensor to meet your needs.

The LV-MaxSonar-EZ Sensors At a Glance

People Detection Wide Beam High Sensitivity	Best Balance	Large Targets Narrow Beam Noise Tolerance
MB1000	MB1010	MB1020
		MB1030
		MB1040

The diagram above shows how each product balances sensitivity and noise tolerance. This does not effect the maximum range, pin outputs, or other operations of the sensor. To view how each sensor will function to different sized targets reference the LV-MaxSonar-EZ Beam Patterns.

Background Information Regarding our Beam Patterns

Each LV-MaxSonar-EZ sensor has a calibrated beam pattern. Each sensor is matched to provide the approximate detection pattern shown in this datasheet. This allows end users to select the part number that matches their given sensing application. Each part number has a consistent field of detection so additional units of the same part number will have similar beam patterns. The beam plots are provided to help identify an estimated detection zone for an application based on the acoustic properties of a target versus the plotted beam patterns.

Each beam pattern is a 2D representation of the detection area of the sensor. The beam pattern is actually shaped like a 3D cone (having the same detection pattern both vertically and horizontally). Detection patterns for dowels are used to show the beam pattern of each sensor. Dowels are long cylindered targets of a given diameter. The dowels provide consistent target detection characteristics for a given size target which allows easy comparison of one MaxSonar sensor to another MaxSonar sensor.

For each part number, the four patterns (A, B, C, and D) represent the detection zone for a given target size. Each beam pattern shown is determined by the sensor's part number and target size.

The actual beam angle changes over the full range. Use the beam pattern for a specific target at any given distance to calculate the beam angle for that target at the specific distance. Generally, smaller targets are detected over a narrower beam angle and a shorter distance. Larger targets are detected over a wider beam angle and a longer range.

People Sensing:

For users that desire to detect people, the detection area to the 1-inch diameter dowel, in general, represents the area that the sensor will reliably detect people.

MB1010

LV-MaxSonar®-EZ1™ Beam Pattern

Sample results for measured beam pattern are shown on a 30-cm grid. The detection pattern is shown for dowels of varying diameters that are placed in front of the sensor

A 6.1-mm (0.25-inch) diameter dowel

B 2.54-cm (1-inch) diameter dowel

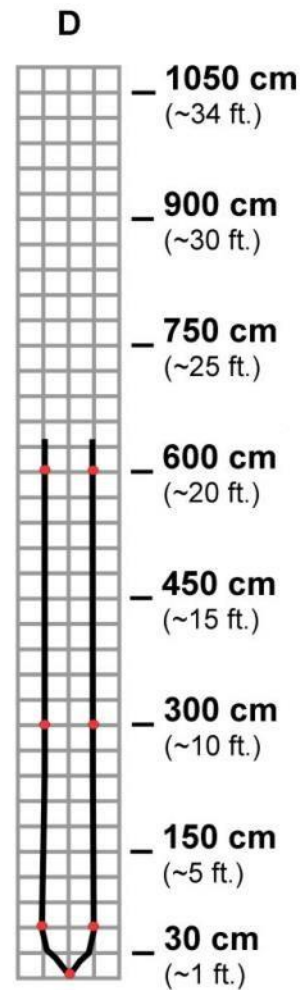
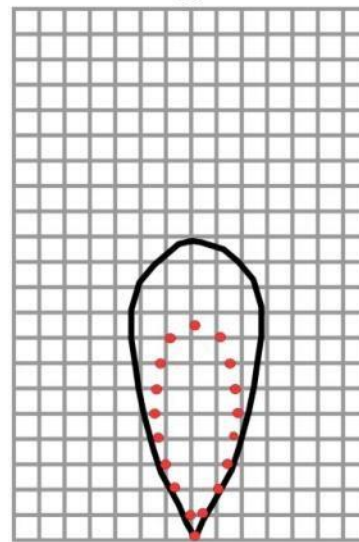
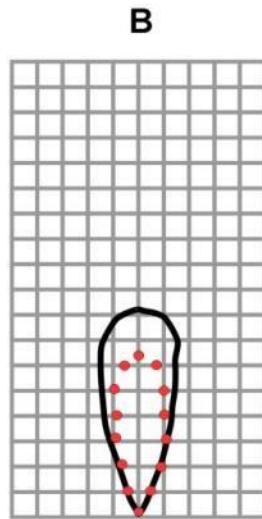
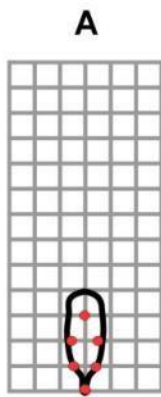
C 8.89-cm (3.5-inch) diameter dowel

D 11-inch wide board moved left to right with the board parallel to the front sensor face.

This shows the sensor's range capability.

Note: For people detection the pattern typically falls between charts A and B.

— 5.0 V
● 3.3 V



Beam Characteristics are Approximate

Beam Pattern drawn to a 1:95 scale for easy comparison to our other products.

MB1010 Features and MB1010 Applications and Benefits

Uses

- Most popular ultrasonic sensor □ Great for people detection
- Low power consumption □ Security
- Easy to use interface □ Motion detection
- Can detect people to 8 feet □ Used with battery power
- Great balance between sensitivity □ Autonomous navigation and object rejection
- Educational and hobby robotics

- Can be powered by many different
- Collision avoidance types of power sources

MB102 -MaxSonar-EZ2

The LV EZ2 is a good compromise between sensitivity and side object rejection. The LV-MaxSonar-EZ2 is an excellent choice for applications that require slightly less side object detection and sensitivity than the MB1010 LV-MaxSonar-EZ1.

LV-MaxSonar

Have the right sensor for your application?

Select from this product list for Protected and Non-Protected Environments.

Protected Environments

	
1 mm Resolution HRLV-MaxSonar-EZ	1 in Resolution LV-MaxSonar-EZ LV-ProxSonar-EZ
	
1 cm Resolution XL-MaxSonar-EZ XL-MaxSonar-AE XL-MaxSonar-EZL XL-MaxSonar-AEL	1 mm Resolution HRUSB-MaxSonar-EZ 1 in Resolution USB-ProxSonar-EZ




Accessories — More information is online.


MB7954 — Shielded Cable

The MaxSonar Connection Wire is used to reduce interference caused by electrical noise on the lines. This cable is a great solution to use when running the sensors at a long distance or in an area with a lot of EMI and electrical noise.



Non-Protected Environments

	
1 mm Resolution HRXL-MaxSonar-WR HRXL-MaxSonar-WRS HRXL-MaxSonar-WRT HRXL-MaxSonar-WRM HRXL-MaxSonar-WRMT HRXL-MaxSonar-WRL HRXL-MaxSonar-WRLT HRXL-MaxSonar-WRLS HRXL-MaxSonar-WRLST SCXL-MaxSonar-WR SCXL-MaxSonar-WRS SCXL-MaxSonar-WRT SCXL-MaxSonar-WRM SCXL-MaxSonar-WRMT SCXL-MaxSonar-WRL SCXL-MaxSonar-WRLT SCXL-MaxSonar-WRLS SCXL-MaxSonar-WRLST 4-20HR-MaxSonar-WR	1 mm Resolution HRXL-MaxSonar-WRC HRXL-MaxSonar-WRCT 1 cm Resolution XL-MaxSonar-WRC XL-MaxSonar-WRCA I2CXL-MaxSonar-WRC
	
1 cm Resolution XL-MaxSonar-WR XL-MaxSonar-WRL XL-MaxSonar-WRA XL-MaxSonar-WRLA I2CXL-MaxSonar-WR	1 cm Resolution UCXL-MaxSonar-WR UCXL-MaxSonar-WRC I2C-UCXL-MaxSonar-WR



F-Option. Available for WR models except UCXL.
For additional protection when necessary in hazardous chemical environments.



MB7950 — XL-MaxSonar-WR Mounting Hardware

The MB7950 Mounting Hardware is selected for use with our outdoor ultrasonic sensors. The mounting hardware includes a steel lock nut and two O-ring (Buna-N and Neoprene) each optimal for different applications.

MB7955 / MB7956 / MB7957 / MB7958 / MB7972 — HR-MaxTemp

The HR-MaxTemp is an optional accessory for the HR-MaxSonar. The HR-MaxTemp connects to the HR-MaxSonar for automatic temperature compensation without self heating.

MB7961 — Power Supply Filter

The power supply filter is recommended for applications with unclean power or electrical noise.

MB7962 / MB7963 / MB7964 / MB7965 — Micro-B USB Connection Cable

The MB7962, MB7963, MB7964 and MB7965 Micro-B USB cables are USB 2.0 compliant and backwards compatible with USB 1.0 standards. Varying lengths.

MB7973 — CE Lightning/Surge Protector

The MB7973 adds protection required to meet the Lightning/Surge IEC61000-4-5 specification.



Remotely Piloted Hovercraft with PID Heading Control

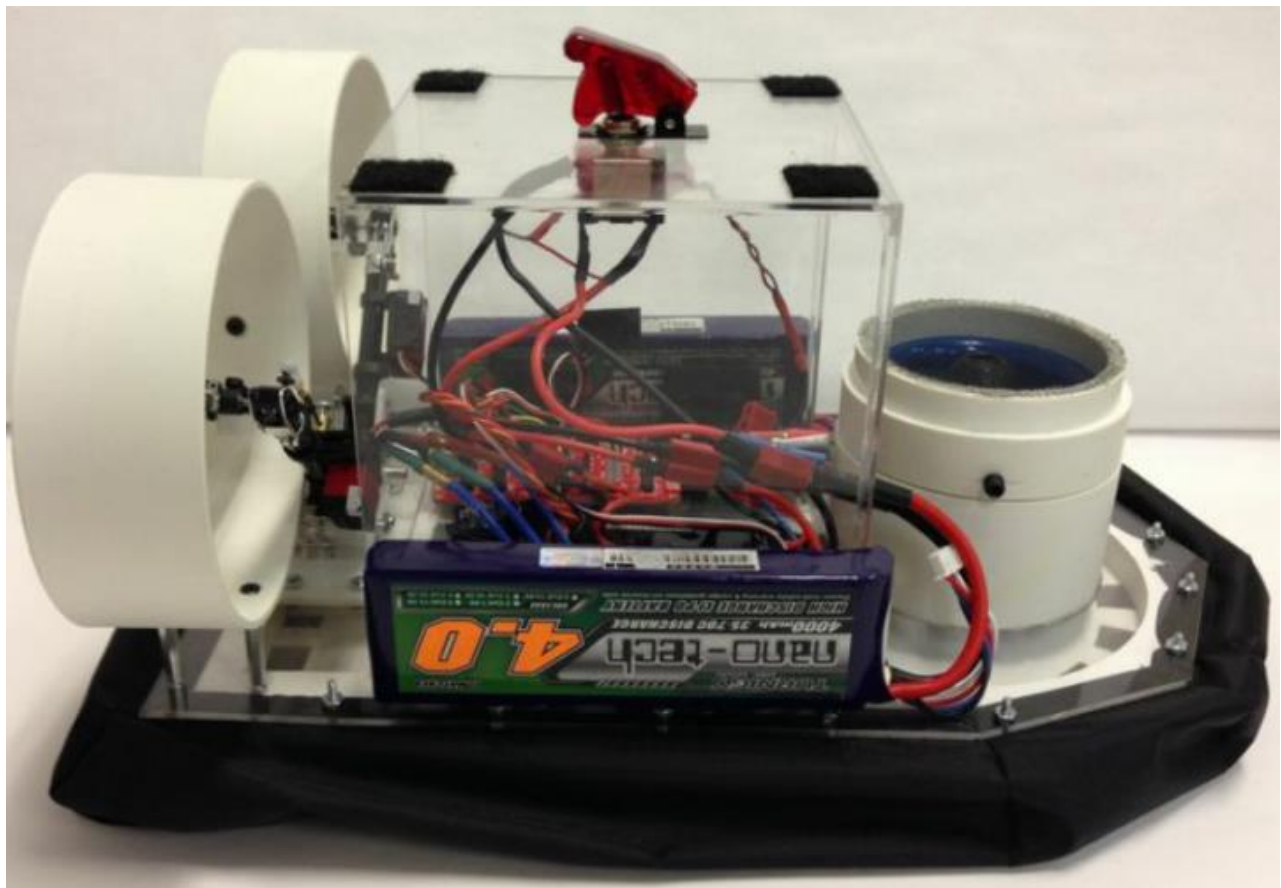
Submission Date:

December 18, 2014

Submitted By:

Cory Hunt & Jose Upegui

For ME445 Final Project



Sponsor: Boeing

Non-Disclosure Agreement: Not required

Executive Summary

The purpose of this project is to design and build a remotely piloted hovercraft (RPH) that can navigate an obstacle circuit with a one-pound payload. The hovercraft is restricted to 18”L x 12”W x 12”H during the time of operation, and the payload must be carried in a container at least 6” above the base of the hovercraft. Additionally the team attempted to implement PID control of the hovercraft’s heading in conjunction with radio control to avoid over-rotating while turning. Due to the complexity and time constraints of the project the team decided to solely implement the PID heading control to demonstrate its ability to maintain a heading.

The project consisted of gathering product requirements from the customer, researching patents and concepts externally, establishing target specifications for a finished product, and finally, generating a concept for alpha prototyping. After receiving the requirements for the RPH from the sponsor and organizing them into target specifications, the team (Boeing-1) was able to come up with a design concept for the first iteration of prototyping. After weighing many ideas in concept generation, the chosen concept was a RPH with a single lift fan and bag skirt, controlled by two rear-mounted variable pitch propulsion fans.

The next step in the design process is to construct an alpha prototype of this chosen design. To start, Boeing-1 will construct a working floating hovercraft base that validates the design for the hovercraft hull and skirt. This prototype will be made from inexpensive materials, and will focus on validating the soundness of air-flow and stability in the base design. The alpha prototype will also be an opportunity to begin simple groundwork programming to control motors and sensors on the RPH.

The final step of the project was to implement the PID heading control in an effort to better control the rate of rotation during turning maneuvers. As the mechanical design required more time than expected, the implementation of the PID control was simplified to include only heading control with no input from the remote receiver. This allowed the design to maintain hovercraft functions while also demonstrating the ability of the PID control to maintain a heading.

Table of Contents

1.0 Introduction	5
1.1 Initial Problem Statement	5
1.2 Objectives	5
2.0 Customer Needs Assessment	6
2.1 Gathering Customer Input	7
2.1.1 Competition Overview/Scoring	7
2.1.2 Course Description	7
2.1.3 Hovercraft Requirements	7
2.1.4 Payload Description	7
2.2 Weighing of Customer Needs	8
3.0 External Search	10
3.1 Patents	10
3.2 Existing Products	11
4.0 Engineering Specification	14
4.1 Establishing Target Specifications	14
4.2 Relating Specifications to Customer Needs	15
5.0 Concept Generation and Selection	17
5.1 Problem Clarification	17
5.1.1 Black Box Model	17
5.1.2 Problem Decomposition Tree	18
5.2 Concept Generation	20
5.2.1 Concept M	20
5.2.2 Concept A	21
5.2.3 Concept U	22
5.2.4 Concept H	23
5.2.5 Prototype Concept	24
5.3.1 Concept Screening	25
5.3.2 Concept Scoring	26
6.0 System Level Design	27
7.0 Special Topics	30
7.1 Preliminary Economic Analyses - Budget and Vendor Information	30
7.2 Project Management	30
7.3 Risk Plan and Safety	30
7.4 Ethics Statement	32
7.5 Environmental Statement	32
7.6 Communication and Coordination with Sponsor	32
8.0 Detailed Design	32
8.1 Manufacturing Process Plan	32
8.1.1 Manufacturing Process Plan	33
8.2 Analysis	37
8.3 Material Selection Process	40
8.4 Component Selection Process	41
8.6 PID Heading Control	43
8.6.1 Mechanical Design Modifications	43
8.6.2 PID Controls	44
8.7 Testing	45
8.7.1 Testing Procedure	45
8.7.2 Testing PID Gains	48
8.7.3 Testing Results Analysis	48
8.7.4 Economic Analysis	52

9.0 Final Discussion	53
9.1 Construction Process	53
9.2 Testing	54
10.0 Conclusions and Recommendations	57
11.0 Self-Assessment (Design Criteria Satisfaction)	58
11.1 Customer Needs Assessment.....	60
11.2 Global and Societal Needs Assessment.....	61
References	62
Appendix A: Project Management.....	63
Appendix B: Full Patent Research	65
Appendix C: Concept Combination Trees	76
Appendix D: Align T-Rex Manual	79
Appendix E: Detailed Computer Aided Design Drawings	83
Appendix F: Complete Bill of Materials	121
Appendix G: Arduino Code RC Controls	122
Appendix H: Arduino Code Self Stabilizer	123

1.0 Introduction

The unique capabilities of improved vehicles work to expand the possibilities of human mobility and travel. The most obvious example of this is the invention of the automobile. The internal combustion engine revolutionized personal travel and brought freedom for people to move about the country with little restriction. More recently even than the automobile, the invention of the airplane has conquered the skies, allowing for even faster and less restrictive travel.

Similarly, the hovercraft presents an interesting set of features that could enhance the extent of human mobility. Just like the automobile and airplane, a hovercraft presents a unique set of capabilities. By hovering just above the ground, a hovercraft can be translated with minimal frictional forces. Additionally, the gap between the vehicle and the travelling surface allow for the traversal of many different types of terrains, including roads, beaches, gravel, and water. Because of this promise of multimodal operation, the dynamics and possibilities of the hovercraft are worth studying and understanding.

However, a disadvantage of hovercrafts is that they are under actuated causing difficult control of their motion. This project seeks to overcome this difficulty by implementing two techniques to improve maneuverability. The first makes use of variable pitch propellers to provide steering through differential thrust. Unlike a traditional design with a ruddered air stream, differential thrust allows for rotational motion with no translational motion as well as the ability to quickly reverse. The second technique applies PID control to the rotation of the hovercraft that prevents the differential thrust from over-rotating during steering.

1.1 Initial Problem Statement

The purpose of this project is to fabricate a working hovercraft to race around an obstacle circuit with a one-pound payload. The obstacle circuit will consist of turns, ramps, and bumps, and will contain a pit stop where the hovercraft can be stopped to offload some of the payload. The hovercraft is restricted to 18"L x 12"W x 12"H during the time of operation, and the payload must be carried in a container at least 6" above the base of the hovercraft. Additionally the hovercraft should be able to return to its original heading position after being applied a small disturbance.

1.2 Objectives

A winning hovercraft will be one that finds a correct balance among the following capabilities. The hovercraft needs to be powerful enough to lift the payload with ease, and robust enough to traverse the obstacles of bumps and ramps in the circuit. The hovercraft needs to be quick in order to complete as many laps as possible. The hovercraft also needs to be mobile enough to successfully complete turns and obstacles unhindered. Software in the vehicle should provide adequate control to the driver to give them a competitive advantage during the race. Since some of these criteria are in conflict with one another, much of the design process will focus on finding an appropriate balance of abilities for the hovercraft. Finally, the hovercraft should use PID control to maintain a stable heading.

2.0 Customer Needs Assessment

To compile a list of needs, Boeing-1 spoke with the corporate sponsors to find a list of project requirements. These requirements and market research of other RPH were turned into needs statements. Boeing-1 used a customer needs hierarchy to determine the overall importance of each need. Then an AHP pairwise comparison table was used to rank the relative importance of each need.

Table 1: Interpreting Customer Statements

Prompt/Question	Customer Statements	Interpreted Needs
Payload Capacity	<ul style="list-style-type: none"> • RPH should be able to carry a one-pound load. • The load should be carried six inches above the RPH deck. • The load can be reduced by 1/5th pound increments during operation. • Load can shift during operation. 	<ul style="list-style-type: none"> • The RPH will support a one-pound load while remaining stable. • The RPH will be secured in a structure standing six inches above the deck. • The load securing system will be adaptable to the changing load. • The RPH will have a container to carry the load • Payload container walls are smooth
Maneuverability	<ul style="list-style-type: none"> • The RPH will be judged by the number of laps through an obstacle course it can complete in fifteen minutes. • The RPH will be tested on a course with obstacles, bumps, slopes, textured surfaces, and possibly water. 	<ul style="list-style-type: none"> • The RPH will be easily maneuverable • The RPH will remain stable and functional in water and on slopes • The RPH must be capable of moving to avoid obstacles • The RPH performs well on carpet and artificial turf • The RPH is capable of going over obstacles
Duration of Operation	<ul style="list-style-type: none"> • The RPH will be tested by fifteen minutes of operation in an obstacle course 	<ul style="list-style-type: none"> • The RPH will be able to operate continuously for at least fifteen minutes.
Size	<ul style="list-style-type: none"> • The RPH should fit within 15" x 12" x 8" 	<ul style="list-style-type: none"> • The RPH will fit within a volume of 15" x 12" x 8" • The payload container fits within a volume of 6" x 6" x 4"
Aesthetics	N/A	<ul style="list-style-type: none"> • RPH has an appealing design.
Control	<ul style="list-style-type: none"> • The hovercraft should be controlled remotely by a team member during competition. 	<ul style="list-style-type: none"> • The hovercraft will be controlled using the Spektrum DX7 controller. • The RPH is controlled using wireless protocol.
Power Storage	<ul style="list-style-type: none"> • Power must be stored in one or multiple Turnigy Nano-tech Li-Po battery packs 	<ul style="list-style-type: none"> • The RPH will store power in at least one of the supplied batteries.
Safety	<ul style="list-style-type: none"> • The RPH should operate safely for both drivers and spectators. 	<ul style="list-style-type: none"> • Propulsion and lift systems will have guarding around all fans. • The RPH must be deemed safe by sponsors prior the race. • Electrical systems have to be safe
Cost	<ul style="list-style-type: none"> • Boeing has allocated \$1000 for Boeing-1 to use. 	<ul style="list-style-type: none"> • Project costs must remain under \$1000 and anything more will be paid out of pocket.

2.1 Gathering Customer Input

Boeing-1 first started gathering the sponsor's needs by studying the description paragraph in the project description found on the "Learning Factory" website. This description provided the general overview that was needed to start general brainstorming about the hovercraft. About one week into the project, the sponsor sent Boeing-1 a formal description of the hovercraft challenge. This document describes how the hovercraft will be scored, information about the obstacle course, and sizing guidelines for the hovercraft.

From these two documents, Boeing-1 was able to come up with some questions to present to the sponsor during weekly meetings. Some questions were regarding the legality of certain devices, namely main lift fans for the hovercraft and methods for carrying the payload. Additionally, during the weekly meetings, Boeing-1 was able to gather the sponsor's feedback on preliminary concepts for the design of the hovercraft. The team mentors gave good advice about design concerns regarding different concepts that Boeing-1 may not have known without prior experience in the field.

2.1.1 Competition Overview/Scoring

The vehicle will carry a 11b payload for 15 minutes. One point will be awarded for every lap completed. The vehicle may stop to offload .11b of payload every lap (.21b will be offloaded on the 1st pit stop only). The pit stop lane will increase the length of a lap completion.

2.1.2 Course Description

The course will consist of a variety of straight sections, turns, and obstacles. Obstacles throughout the course may consist of ramps with a max slope of 20 degrees and .5in tall obstacles. Any other obstacles can be driven around. The course will likely be a carpet or astro-turf material.

2.1.3 Hovercraft Requirements

The vehicle shall be controlled using a wireless protocol. The vehicle must operate continuously for 15 minutes. A Turnigy 4000mAh 3S 35~70C Lithium Polymer (LiPo) battery pack is provided. The vehicle base refers to the base structure of the vehicle that the skirt attaches to. This is the basis for the measurement of the payload height. The base of the hovercraft may not exceed 8in wide. The vehicle with the skirt deployed/fully inflated must not exceed 18"Lx12"Wx12"H. The vehicle must be considered safe by the team sponsors prior to the race. This includes, but is not limited to, proper covering and security off all wires, vehicle free of FOD (foreign object debris), and props protected by a shroud or covering (must stay on during the race). The hovercraft has to maintain a stable heading using PID control and gyro/accelerometer feedback.

2.1.4 Payload Description

The payload will consist of 11b of a material that is easily removable to allow for quick payload reduction during pit stops. The payload may consist of multiple pieces that may shift during the race. The vehicle shall have an open container with internal dimensions of 6"Lx6"Wx4"H. The container must have smooth surfaces as to not prohibit the movement of any payload placed within. The base of the container must be set 6in above the base of the vehicle. The 4in height of the container does not apply to the 12in height restriction imposed upon the vehicle.

2.2 Weighing of Customer Needs

The first step in weighing customer needs was to compile, group and rank the needs qualitatively. Boeing-1 achieved this by completing the customer needs hierarchy chart seen below in Table 2 which provides categories and the relative importance of each need that was later used in the pairwise comparison chart.

Table 2: Customer Needs Hierarchy

Payload Capacity		Maneuverability	
***	The RPH will support a 1 pound load while remaining stable.	***	The RPH will be easily maneuverable
**	The Payload will be secured in a structure standing 6 inches above the deck.	**!	The RPH will remain stable and functional in water and on slopes
*!	The load securing system will be adaptable to the changing load.	***	The RPH must be able to move to avoid obstacles
		***	The RPH is Capable of going over obstacles
***	The RPH will have a container to carry the load	***	The RPH performs well on carpet and AstroTurf
**	Payload container walls are smooth		
Size		Duration of Operation	
***	The RPH will fit within a volume of 15in x 12in x 8in	***	The RPH will be able to operate continuously for at least 15 minutes.
***	The payload container fits within a volume of 6in x 6in x 4in		
Aesthetics		Control	
*!	RPH has an appealing design.	**	The RPH will be controlled using the Spectrum DX7 controller.
		***	The RPH is controlled using wireless protocol
Power Storage/Distribution		Safety	
		**	Propulsion and lift systems will have guarding around all fans
*	The RPH will store power in at least one of the supplied batteries.	**	The RPH has to be considered Safe by Sponsors prior the race
Cost		**	Electrical systems have to be safe
***	Project costs must remain under \$1000.		

Hierarchical list of primary and secondary customer needs for the Boeing Hovercraft. Importance ratings are dictated by the number of * next to the need statement. ! denotes a latent need

The categories found in the hierarchical needs chart were turned into evaluation criteria to be used to judge the customer needs. The next step was to determine the relative importance of each of these criteria. By comparing each of the criteria against each other, the weighed value of each was

found. The pairwise comparison chart was used for this process and it is shown below in Table 3 and the results were summarized in Figure 1.

Table 3: AHP Pairwise Comparison Chart to Determine Criterion Relative Weights

Description	Payload Capacity	Maneuverability	Duration of Operation	Size	Aesthetics	Control	Power Storage	Safety	Cost	Total Score	Weighted Value
Payload Capacity	1.00	0.50	1.00	3.00	5.00	0.25	0.33	4.00	3.00	18.08	0.13
Maneuverability	2.00	1.00	2.00	3.00	5.00	1.00	4.00	4.00	4.00	26.00	0.18
Duration of Operation	1.00	0.50	1.00	4.00	5.00	0.33	5.00	4.00	4.00	24.83	0.17
Size	0.33	0.33	0.25	1.00	5.00	0.50	3.00	2.00	1.00	13.42	0.09
Aesthetics	0.20	0.20	0.20	0.20	1.00	0.20	0.50	2.00	2.00	6.50	0.05
Control	4.00	1.00	3.00	2.00	5.00	1.00	5.00	4.00	5.00	30.00	0.21
Power Storage	3.00	0.25	0.20	0.33	2.00	0.20	1.00	0.50	0.25	7.73	0.05
Safety	0.25	0.25	0.25	0.50	0.50	0.25	2.00	1.00	3.00	8.00	0.06
Cost	0.33	0.25	0.25	1.00	0.50	0.20	4.00	0.33	1.00	7.87	0.05
Total										142.43	0.99

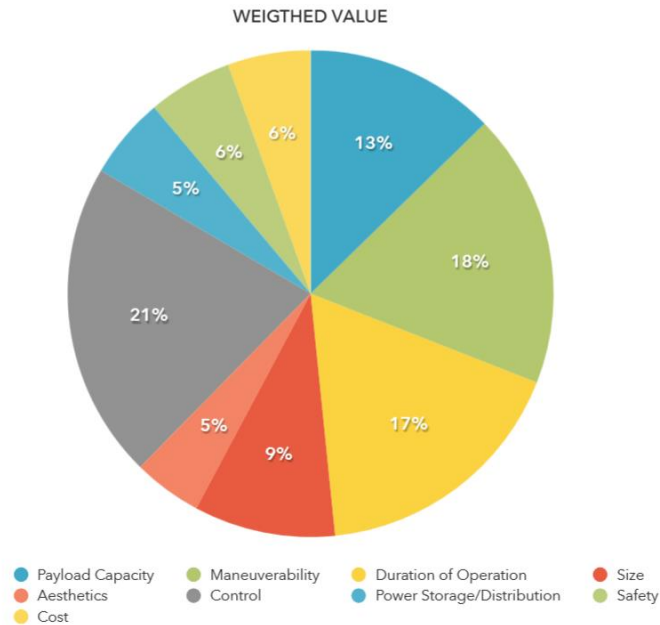


Figure 1: AHP Weighted Values

3.0 External Search

To better understand the systems that are needed in a hovercraft, Boeing-1 researched materials, control systems, propulsion systems and skirt designs. This research was done by finding patents and other literature about these specific topics. By understanding common solutions in these problem areas, Boeing 1 can meet the customer's needs more quickly and effectively while maintaining an ethical standard of academic integrity.

Hovercrafts are vehicles designed to travel over various terrains on a cushion of air. A hovercraft is able to float a small distance above the travel surface using fans to pressurize a volume of air below the craft, surrounded by a flexible skirt. The skirt usually has vents, or is completely open at the bottom, allowing the pressurized air to escape, lifting the craft off the travel surface onto a cushion of air. By elevating the craft above the travel surface without the use of wheels, translation movement can be achieved with minimal frictional costs.

In order to control the craft transnationally, there are two common methods. One common method of propulsion is using a single fan with rudders to steer the craft. As the single fan produces thrust and air-flow across the rudders in the back, they become usable as control surfaces to steer the vehicle. The other common method uses two fans in the rear of the craft placed side by side. A difference in thrust between the two fans generates a torque to turn the craft.

Typically, a small-scale toy hovercraft (similar in usage and scale to the RPH that Boeing-1 will be producing) is controlled remotely by some form of handheld R/C controller. Controls are normally fairly simple, and control the relative speeds of the rear fans or the position of the back rudder.

3.1 Patents

Patent research can be a very valuable tool in the design process. For Boeing-1 it not only revealed some design concepts that would prove valuable in concept generation, but also it highlighted other concepts that would be clear violations of current patents.

The search for existing patents was conducted with the primary goal of finding current products' solutions to both hovercraft stability as well as vehicle steering and directional control. The full search results can be seen in Appendix C but the summarized art-function matrix is shown below. What Boeing-1 discovered was that a system of multiple rotatable thrust fans would clearly infringe on patent 6591928 but either a movable ballast, or adjustable plenum system for pitch/roll control was viable as their respective patents had expired.

Table 4: Art-Function Matrix

Function	Art						
	Movable Ballast	Adjustable Plenum	Multiple Rotatable Thrusters	One Steering Fan + One Thrust/Lift Fan	Two fixed Lateral Opposed Fans w/ Rudders	Dual Air Cushion	Dual Thrust Fans
Pitch/Roll Stability	3589058	3279553					
Steering/ Propulsion			6591928	7032698	20130333968		
Aesthetics						564046	451560

3.2 Existing Products

Boeing-1 searched for remotely piloted hovercrafts currently on the market to evaluate different designs and to find if any would be capable of traversing an obstacle course while carry the one-pound payload. There were 4 products evaluated that are currently on the market. A description of the features of these products can be seen in Table 5 below. These RPH were rated on their ability to meet the requirements of the project in Table 6. Each RPH was rated on a scale of 1 (poor) to 5 (excellent) on their ability to meet each of the customer needs. The results of the search showed that no products exist that are suitable to meet all the requirements of the Boeing challenge. All of these systems utilize rubber skirts and fans to provide propulsion and lift. Some use multiple rear fans to propel and steer the craft where others use a single fan and rudders. None of these fans use any type of system to control the amount of lift and only one of these has a button that will cut the lift. None of these systems have the control necessary to traverse an obstacle course. None of these systems have any storage area where they could carry the required one-pound weight.

Boeing-1 conducted a search for existing control schemes for a hovercraft. There are many academically oriented papers that study the dynamics of hovercraft similar to what Boeing-1 proposes to build. Contained in these are simplifying assumptions and equations that govern the control dynamics of the hovercraft from a software and electrical systems perspective. Additionally, Boeing-1 discovered an academic paper that describes a positional tracking scheme used on a small remote control hovercraft. Knowledge in these problem areas will be valuable as Boeing-1 plans the capabilities for the RPH. Since these papers are public documents, information from them that is used in the creation of Boeing-1's RPH will be cited and credited to the appropriate authors.

Table 5: Features of Existing Products

Product:	Features
<p>Microgear BW-777</p> 	<ul style="list-style-type: none"> ● 6V 800 MAH NICD battery ● Rubber skirt ● Single rear fan ● Rudders
<p>Zhi Lun 6653</p> 	<ul style="list-style-type: none"> ● 2 sectioned rubber skirt ● Independent fan control ● 9.6V 700mAh battery ● Max speed 20 km/h on ground
<p>The Straightway</p> 	<ul style="list-style-type: none"> ● 9-13 minute run time ● 2 fan propulsion ● Rubber skirt
<p>Speed Boat</p> 	<ul style="list-style-type: none"> ● 3 section rubber skirt ● 2 motors provide lift ● Lift cut button included ● 9.6V 700 mAh battery

Table 6: Relating Customer Needs to Existing Products

The following chart describes how the different products in the market meet the set requirements for the desired hovercraft.

No		Need	Microgear	Zhi Lun 6653	The Straightway	Speed Boat
1	Payload	The RPH will support a 1 pound load while remaining stable.	2	3	2	2
2	Payload	The Payload will be secured in a structure standing 6 inches above the deck.	1	1	1	1
3	Payload	The load securing system will be adaptable to the changing load.	1	1	1	1
4	Payload	The RPH will have a container to carry the load	1	1	1	1
5	Payload	Payload container walls are smooth	1	1	1	1
6	Size	The RPH will fit within a volume of 15in x 12in x 8in	5	5	5	5
7	Size	The payload container fits within a volume of 6in x 6in x 4in	1	1	1	1
8	Aesthetics	RPH has an appealing design.	3	4	4	3
9	Aesthetics	Instills pride	2	3	2	2
10	Power Distribution	The RPH will store power in at least one of the supplied batteries.	1	1	1	1
11	Cost	Project costs must remain under \$1000.	NA	NA	NA	NA
12	Maneuverability	The RPH will be easily maneuverable	2	3	3	2
13	Maneuverability	The RPH will remain stable and functional in water and on slopes	3	3	3	3
14	Maneuverability	The RPH must be able to move to avoid obstacles	2	3	2	2
15	Maneuverability	The RPH is Capable of going over obstacles	3	2	1	2
16	Maneuverability	The RPH performs well on carpet and astro-turf	3	3	3	3
17	Operation	The RPH will be able to operate continuously for at least 15 minutes.	1	2	2	1
18	Control	The RPH will be controlled using the Spektrum DX7 controller.	1	1	1	1
19	Control	The RPH is controlled using wireless protocol	1	1	1	1
20	Safety	Propulsion and lift systems will have guarding around all fans	5	5	5	5
21	Safety	The RPH has to be considered Safe by Sponsors prior the race	5	5	5	5
22	Safety	Electrical systems have to be safe	5	5	5	5

4.0 Engineering Specification

4.1 Establishing Target Specifications

Team Boeing-1 created a list of measureable metrics to evaluate the desired engineering specifications. Table 7 below shows the metrics, their relative importance as well as marginal and ideal values for these metrics.

Table 7: Target Specifications

Metric #	Need #	Metric	Imp .	Units	Marginal Value	Ideal Value
1	1,4	Payload container can hold at least 1 pound	5	lb	1	5
2	2,4	Payload container is 6 in above the deck	5	in	8	6
3	3,4,12	Payload container allows for easy removal of cargo	4	s	2	1
4	3,4	Payload container is water-tight	2	in ³ /s	1	0
5	3,4,5	Payload container is made of smooth material	4	microns	0.12	0.03
6	3,4,7	Payload container is at least 6in x 6in x 4in	5	in	6 x 6 x 4	6.25 x 6.25 x 4.25
7	6	RPH dimensions doesn't exceed 18in x 12in x 12in	5	in	18 x 12 x 12	12 x 10 x 10
8	6,12	RPH base may not exceed 8in in width	5	in	8	8
9	6,15	Maximum inflated Skirt Thickness	2	in	5	1.5
10	6,12	Maximum base length	3	in	18	12
11	6,1	Maximum base Thickness	3	in	1	0.25
12	8,9,21	RPH instills pride through design features	1	yes/no	no	yes
13	10	Power is stored in at least one electrical battery	2	unit	3	1
14	11	RPH development cost is at most 1000 USD	4	USD	1000	500
15	12	New user can learn to drive the RPH within 15 minutes	3	min	15	1
16	13	RPH can operate on water	4	mi/h	5	15
17	12,14	RPH remains operational after going over obstacles	4	yes/no	yes	yes
18	12,16	RPH performs on carpet and astro-turf surfaces	4	mi/h	10	20
19	12	RPH has a sharp radius turn radius	4	in	24	0
20	12	RPH battery is changed fast	4	s	60	5
21	12	RPH battery/payload can be changed without use of tools	5	yes/no	no	yes
22	12,13	RPH doesn't flip over when going over 20 degree slopes	5	yes/no	*stationary	yes
23	17,12	RPH runs for at least 15 minutes none stop	4	yes/no	*battery change	yes
24	18	The RPH uses the Spektrum DX7 controller	3	yes/no	yes	yes
25	19	The RPH is controlled using wireless protocol	5	yes/no	yes	yes
26	20,21	Dangerous mechanical components have safety features	3	yes/no	no	yes
27	20,21,9,8	The RPH looks safe	3	yes/no	no	yes
28	9,8,21	Wires are hidden from view during operation	2	% of wire exposed	100	10
29	21,22	Wire connections have been insulated	3	yes/no	no	yes
30	12	The RPH can come to a stop quickly	4	s	10	1

4.2 Relating Specifications to Customer Needs

The task of developing need statements and associated metrics can be confusing. It can be challenging to be certain that each customer need is satisfied by at least one metric. Boeing-1 used a Needs Metrics Matrix, shown below in Table 8, as a visual aid to support the decision-making process. Table 8 shows all of the solutions Boeing-1 developed for each sub problem in the design of the hovercraft. Appendix D shows how these concepts were used to create each of Boeing-1's initial design concepts. Table 9 shows that each of Boeing-1's customer needs is satisfied by at least one of these metrics.

Table 8: Explore Concepts Systematically

Solutions To Sub problem Of Steering The RPH	Solutions To Sub problem Of Skirt Design	Solutions To Sub problem Of Payload
<ul style="list-style-type: none"> - Differential speeds on propulsion fans - Rotating propulsion fans - Rudders - Directional fans on the sides - One fan facing each direction of motion - Rotating Mass - Fan pattern 	<ul style="list-style-type: none"> - Fingers - Multiple skirts - Toe nails (friction free material on skirt) - Bag and fingers - Bag skirt - Doughnut Skirt - Protective Material added to skirt - Pre-inflated Skirt - Irregular Shape Skirt 	<ul style="list-style-type: none"> - Mass supported by springs - Reverse Pendulum - Threaded road and motor 1D - Threaded road and motor 2D - Belt system and motor 1D - Belt system and motor 2D - Fixed payload - Threaded road and rotating disk
Solutions To Sub problem Of Fan Position	Solutions To Sub problems Of Body Design	Solution to Sub problem of Propulsion-Braking
<ul style="list-style-type: none"> - Front - Middle - Back - Front-Back - Front-Middle - Middle-Back - Front-Middle-Back 	<ul style="list-style-type: none"> - Square - Circular - Triangular - Rounded front - Rounded front and back - Laminated sheets base - Solid body - Rounded corners 	<ul style="list-style-type: none"> - Hinges for braking - Fan blades reversible direction - Nozzle - Use air from skirt - Compressed air tanks - Stacked fans for differential speed - Variable pitch propeller - Differential speeds fans - Fast skirt deflation mode
Solutions To Sub problem Of Inflation		
<ul style="list-style-type: none"> - Steam - Chemical - Unique Fan - Multiple Fans - From Propulsion Fans 		

Table 9: Needs-Metrics Matrix

[illegible]

5.0 Concept Generation and Selection

5.1 Problem Clarification

In order to successfully continue developing concepts to solve customer needs and meet the target specifications, the problem must be clearly defined. Boeing-1 implemented two different methods, the first is the black box model and the second is a problem decomposition tree.

5.1.1 Black Box Model

At the beginning of a design project the problem can be vague and difficult to understand. One method of clarifying the problem is a Black Box Diagram, shown for Boeing-1 below. This method clearly shows the inputs and the outputs of the designed system, which is represented as a black box. The box is then expanded and interior components are included. By showing how the components interact, Boeing-1 can better generate concepts that will accomplish the required tasks.

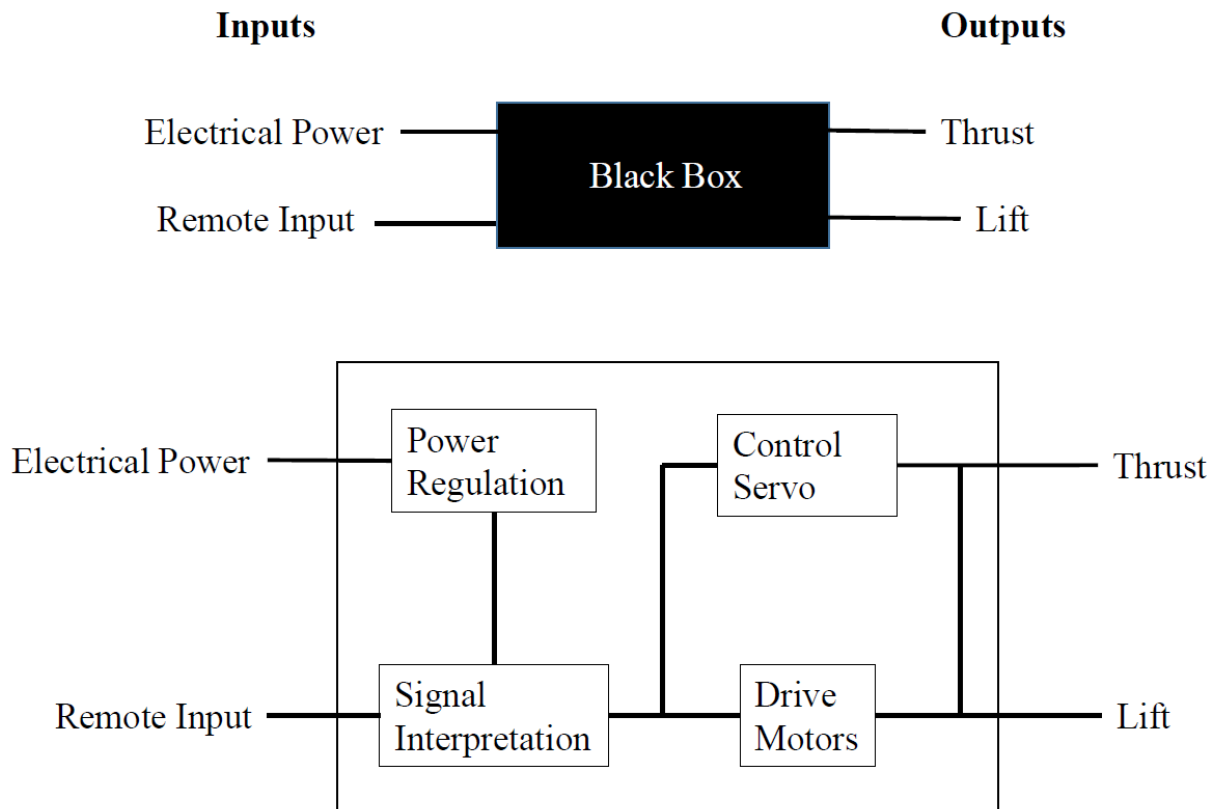


Figure 2: Black Box Diagram

5.1.2 Problem Decomposition Tree

The Figure below provides a visual map that breaks down the hovercraft's six main functions into sub-functions that must be accomplished to satisfy the specifications that were previously developed from the customer needs. The map identifies which sub-functions require further concept generation. Some sub-functions such as the energy storage (Li-Po battery) and the plenum were either already provided by the customer or a solution had already been decided upon by the group unanimously and no further concepts were required. Other sub-functions had no clear solution and required brainstorming and concept generation/selection. These sub-functions are identified on the map below with a box containing the word "yes" under the column "Concept Generation".

The first problem, "store/deliver power" was aptly broken into sub-functions for storing power and delivering power. These two sub-functions required little development time as Li-Po batteries provided by the customer and the electrical power provided by the batteries is most easily transported by conductive metal wires.

The second problem relates to the source of floatation needed by the hovercraft to minimize friction with the ground surface. The function was divided into four sub-functions of creating, dispersing, entrapping, and altering the air-flow under the hull of the hovercraft. Research of hovercraft construction and design principles revealed that the air-flow, as power is electrical, would best be produced by an electric motor-powered fan. A plenum redirects the air-flow into the skirt, which entraps it to create a cushion of high pressure air that lifts the hovercraft. Since the lift fan/motor only required sizing and the plenum design was unanimously agreed upon by the group, neither sub-function required further concept generation. However; the skirt and floatation adjustability were not well understood and required more research as well as concept generation and selection.

The third primary function shown in green on the map relates to thrust generation for propulsion of the hovercraft. This function was divided into two sub-functions of producing and altering the thrust force. The group understood that their power storage limited the design to using an electric motor to turn a fan/propeller. The means to alter the thrust force, however, could be accomplished through many different means. Further concept generation was required for this sub problem, shown by a "yes" under the "Concept Generation" column of the map below.

The fourth main function is closely related to the third in that it identifies the sub-functions related to the direction of thrust force generated. The first sub-function is repeated and the second identifies the need for a moment to be imposed on the hovercraft in order for directional changes to be accomplished. Concept generation was required for the second sub-function as many methods for imposing a moment were discussed within team design meetings. Concepts would then be scored and ranked.

The fifth main function identifies different sub-functions relating to the main supporting structures of the hovercraft. The sub-functions of chassis, fastening components, and removing/replacing components were condensed into two means. The chassis must be constructed of strong, lightweight material that can be machined. This problem warranted concept generation as many diverse materials could be used to accomplish the same sub-function. The two sub-functions related to fasteners were both solved by the same means of using reusable lightweight fasteners

but required concept generation to determine the exact method.

The last main function relates to methods in which the payload required by the customer would be supported. Sub-functions include a 6” platform to elevate the payload, a means to secure the payload, and a means to alter the payload support to accommodate different size loads. The first sub-function only required a lightweight structure that did not require concept generation while the second two sub-functions initially required further concept generation until the customer fully defined the need in a weekly meeting.

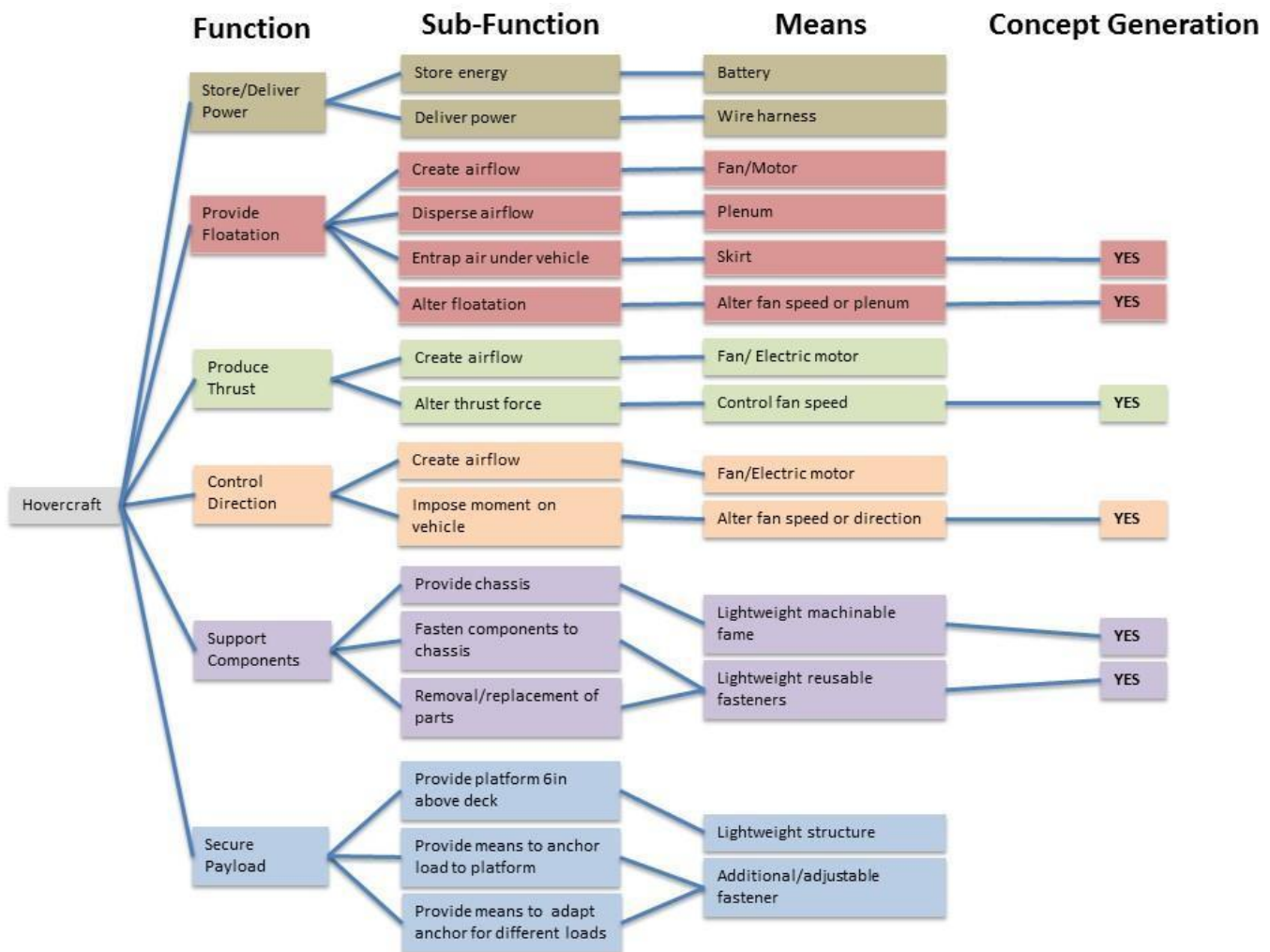


Figure 3: Problem Decomposition Tree

5.2 Concept Generation

During concept generation, each team member of Boeing-1 sketched three unique ideas for any function of the hovercraft. Each member then expanded each other member's concept to generate a new concept. This process was repeated until a total of 48 concepts were sketched. These ideas were then inventoried, and the different functions of each concept were isolated and listed. With all of the brainstormed concepts listed for every function, each team member selected a combination of the concepts to constitute a full concept for the hovercraft. See concept combination trees in Appendix D, which visually connect the concepts from the systematic exploration shown in section 4.2.

5.2.1 Concept M

Concept M is a fairly traditional hovercraft design with one lift fan and two back propulsion fans for steering and thrust. The two propulsion fans in the rear of the vehicle turn the craft by spinning at different speeds. The skirt for this design is a simple bag skirt. In order to carry the payload, this design uses a statically positioned payload container.

The main focuses of this design are simplicity and manufacturability. By sticking to a tried and true method, this concept has a high likelihood of working as designed. Additionally, the simplicity of each component in the system would cut down on manufacturing and design time, providing more time for tuning, programming, and practice.

Table 10: Concept M Characteristics

Power Storage	Li-Po Battery
Floatation	Bag skirt and single lift fan with plenum
Propulsion	Two rear-mounted fans
Steering	Thrust differential between rear facing thrust fans
Structure	Lightweight body and chassis assembly
Payload	Statically mounted payload

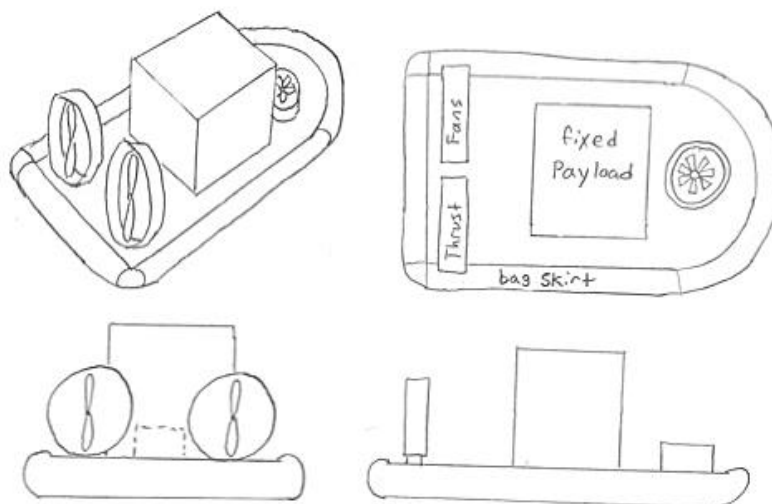


Figure 4: Concept M

5.2.2 Concept A

Concept A is a non-traditional hovercraft design that uses four directional thrust fans as well as a main lift fan for propulsion. Four symmetrically placed propulsions fans provide the ability to thrust in any direction at any time by varying the speeds of the thrust fans. Similar to concept M, this design also utilizes the bag skirt and stationary payload system for simplicity.

The focus of this design is maneuverability. This hovercraft has the advantage of being able to travel in any direction regardless of its orientation. This essentially eliminates the front from the vehicle and allows for precise cornering and braking. The difficulties with this vehicle would stem from its ability to be rotated in any direction. A rate gyro would have to be utilized to make the craft controllable, putting more complexity in the programming.

Table 11: Concept A Characteristics

Power Storage	Li-Po Battery
Floatation	Bag skirt and single lift fan with plenum
Propulsion	Four symmetrically mounted thrust fans
Steering	Thrust between four fans can produce omnidirectional movement
Structure	Lightweight body and chassis assembly
Payload	Statically mounted payload

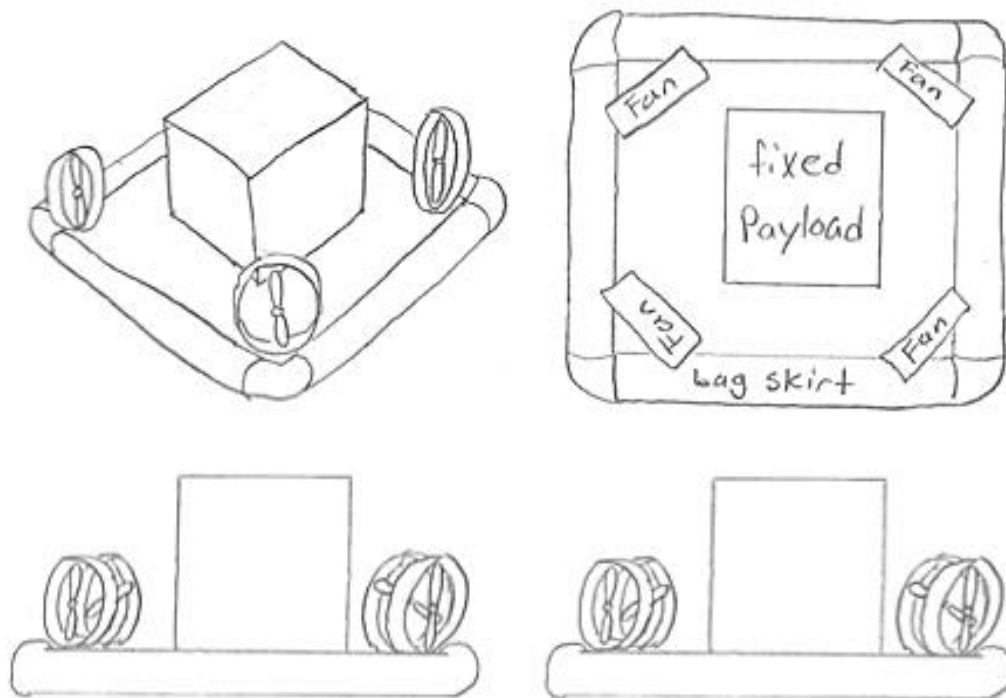


Figure 5: Concept A

5.2.3 Concept U

Concept U utilizes a main lift fan and two actuated propulsion fans in the back. The two fans in the back would be actuated in two ways. Firstly, the thrust fans would swivel at the base, producing thrust lateral to the orientation of the craft. Secondly, these rear fans would have variable pitch blades to allow for quick thrust reversal and braking. The skirt system would be a finger skirt that would allow for better rough terrain traversal. The payload would be mounted on a movable ballast system in order to dynamically shift weight for maneuverability.

The advantage of this concept is that it keeps a traditional shape while providing advanced control and stability features. The variable pitch directional propulsion fans would provide great control for the craft, while the movable ballast and finger skirt system would ensure that the craft could traverse obstacles effectively. The disadvantages to this craft are in its complexity. Many of the components are complicated pieces of machinery that would take a long time to design and manufacture. Additionally, integrating these multiple actuators would ramp up the programming complexity.

Table 12: Concept U Characteristics

Power Storage	Li-Po Battery
Floataction	Finger skirt and single lift fan with plenum
Propulsion	Two rear-mounted fans with variable pitch and swivel
Steering	Thrust differential between rear facing thrust fans, as well as laterally movement due to fan swivel.
Structure	Lightweight body and chassis assembly
Payload	Moveable ballast system to increase maneuverability

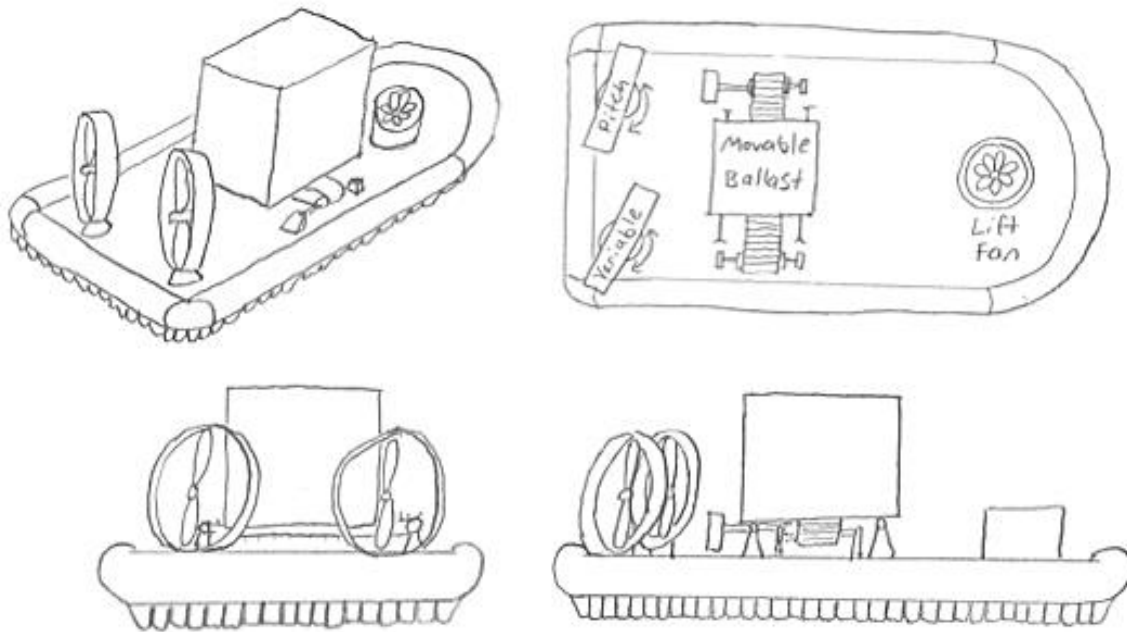


Figure 6: Concept U

5.2.4 Concept H

Concept H is a two-fan design that uses one fan for lift and another single fan for propulsion. A rudder system in the rear would allow the craft to change direction. Additionally, the rear fan would be equipped with variable pitch blades, allowing for which reversal of thrust and braking. The skirt design is a finger skirt designed for obstacle traversal, and the payload would be carried on a movable ballast system.

This design allows for effective moment with only two fans. The finger skirt allows for better obstacle traversal, and the movable ballast system would allow for a more maneuverable craft. Similar to concept U, the disadvantages of this concept are in its complexity. The finger skirt and the movable ballast adds additionally manufacturing complexity to the craft and would be require a lot of work in programming in order for the systems to produce an advantage.

Table 13: Concept H Characteristics

Power Storage	Li-Po Battery
Floatation	Finger skirt and single lift fan with plenum
Propulsion	Single rear-mounted fan
Steering	Rudder system attached to the rear fan, as well as variable pitch blades for thrust reversal and braking
Structure	Lightweight body and chassis assembly
Payload	Movable ballast system

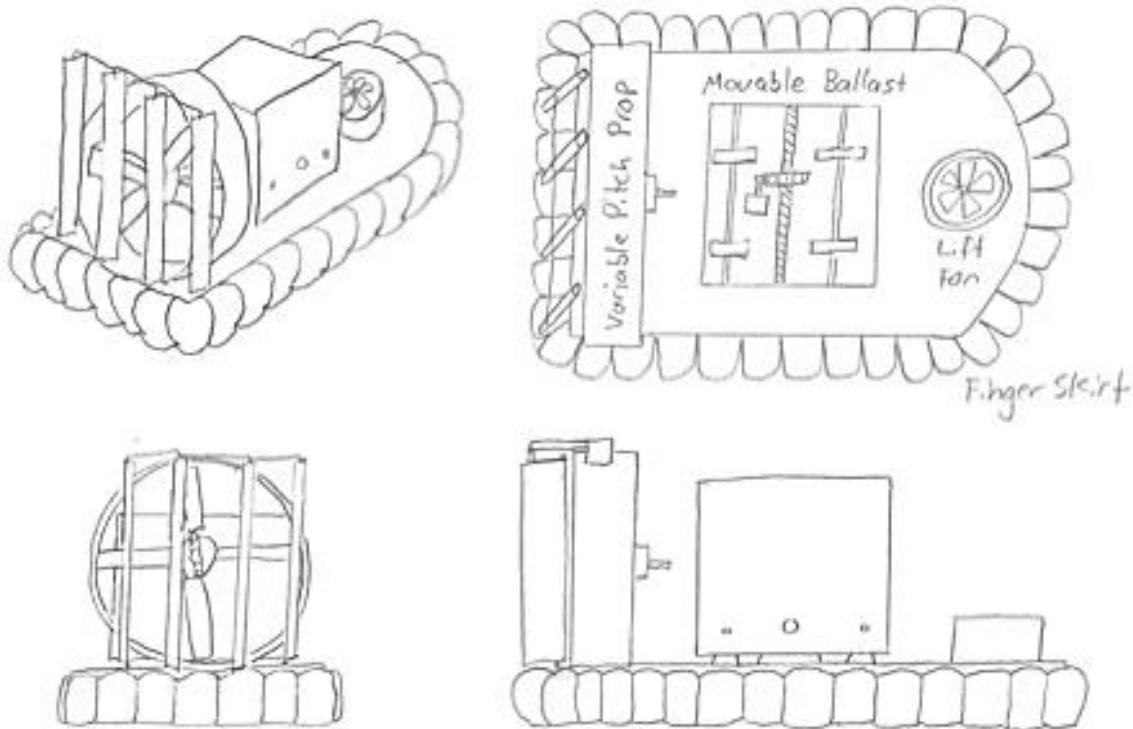


Figure 7: Concept H

5.2.5 Prototype Concept

The prototype concept was agreed upon as an effective combination of the valuable insights of the previous four concepts. This concept features a two-fan system, one for thrust and one for lift. The rear mounted thrust fan would be controlled via rudder and variable pitch, allowing the craft to steer and brake effectively. The skirt would be a bag skirt inflated by a lift fan and plenum system. The payload would be statically mounted.

The advantage of this design is that it can provide adequate maneuverability while still retaining a relatively simple design. The rear fan is the most complex portion of the concept, but this complexity has the potential to pay off with the greatest gains in maneuverability. The bag skirt and statically mounted payload keep the design simple and easily manufactured within the time constraint.

Table 14: Prototype Concept Characteristics

Power Storage	Li-Po Battery
Floatation	Bag skirt and single lift fan with plenum
Propulsion	Single rear-mounted fan with rudder and variable pitch systems
Steering	Rudder system of rear thrust fan
Structure	Lightweight body and chassis assembly
Payload	Statically mounted payload

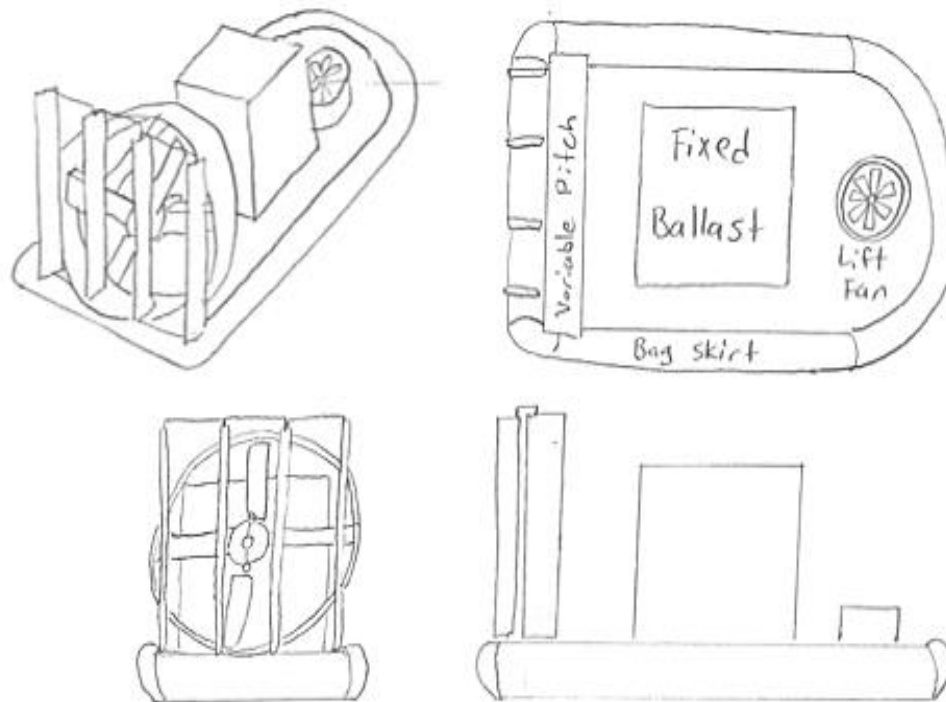


Figure 8: Prototype Concept

5.3 Concept Selection

5.3.1 Concept Screening

The first process in systematically organizing the concepts was concept screening. The purpose of concept screening was to determine whether or not a concept showed more or less effectiveness than a baseline RPH. The difference in effectiveness across many metrics was noted by a '+', '0', or '-' for each concept. A '+' indicates that the concept has considerable advantage in the metric. A '0' indicates that the concept has no perceivable advantage in the metric. A '-' Indicates that the concept is considerably less advantageous in the metric.

Table 15: Concept Screening Matrix

Concepts					
Selection Criteria	M	H	A	U	Microgear
Controllability	+	+	+	+	o
Complexity	o	-	-	-	o
Cost	-	-	-	-	o
Manufacturability	+	-	-	-	o
Durability	o	+	+	+	o
Serviceability	+	+	+	+	o
Power Usage	-	-	-	-	o
Stability	o	+	-	+	o
Safety	o	o	o	o	o
Ease of Use	o	+	+	+	o
Aesthetics	o	+	-	+	o
Sum +'s	3	6	4	6	o
Sum 0's	6	1	1	1	10
Sum -'s	2	4	6	4	o
Net Score	1	2	-2	2	0
Rank	2	1	5	1	4
Continue?	yes	combine	no	combine	no

5.3.2 Concept Scoring

After the concepts were screened for their effectiveness in different metrics, the metrics were weighted by importance as a percentage, with all metrics summing to 100%. Then, each concept was scored from 1-5 on each metric. Multiplying the score by the metric weight and summing across all metrics yields a score for the concept. This process helped eliminate favouritism in concept selection and allowed Boeing-1 to select the concept that was most promising in producing an effective and competitive RPH.

Table 16: Concept Scoring Matrix

Selection Criteria	Percentage	Concepts					
		M		H		U	
		Rating	Weighted Score	Rating	Weighted Score	Rating	Weighted Score
Controllability	15%	3	0.45	4	0.6	4	0.6
Complexity	10%	4	0.4	2	0.2	1	0.1
Cost	5%	4	0.2	2	0.1	2	0.1
Manufacturability	13%	5	0.65	3	0.39	2	0.26
Durability	10%	4	0.4	5	0.5	5	0.5
Serviceability	4%	4	0.16	4	0.16	4	0.16
Power Usage	9%	4	0.36	3	0.27	3	0.27
Stability	15%	2	0.3	4	0.6	5	0.75
Safety	5%	4	0.2	4	0.2	4	0.2
Ease of Use	12%	2	0.24	4	0.48	4	0.48
Aesthetics	2%	1	0.02	5	0.1	5	0.1
Total Score	100%	3.38		3.6		3.52	
Rank		3		1		2	
Continue?		no		Combine		Combine	

Concept M: This concept was deemed to be considerably simpler than other concepts. Its ease of manufacturing and general simplicity were its biggest advantages. The concept's disadvantages were also due to its simplicity, as it was deemed to be not as easily controllable.

Concept H: This concept scored the highest because of its ease of control and stability. The concept has more inherent complexity than concept M, but this complexity is warranted for the extra performance.

Concept U: This concept was the most complex of the three. It offered significant improvements in stability and controllability over concept M, though the added complexity was unwarranted as compared to concept H.

6.0 System Level Design

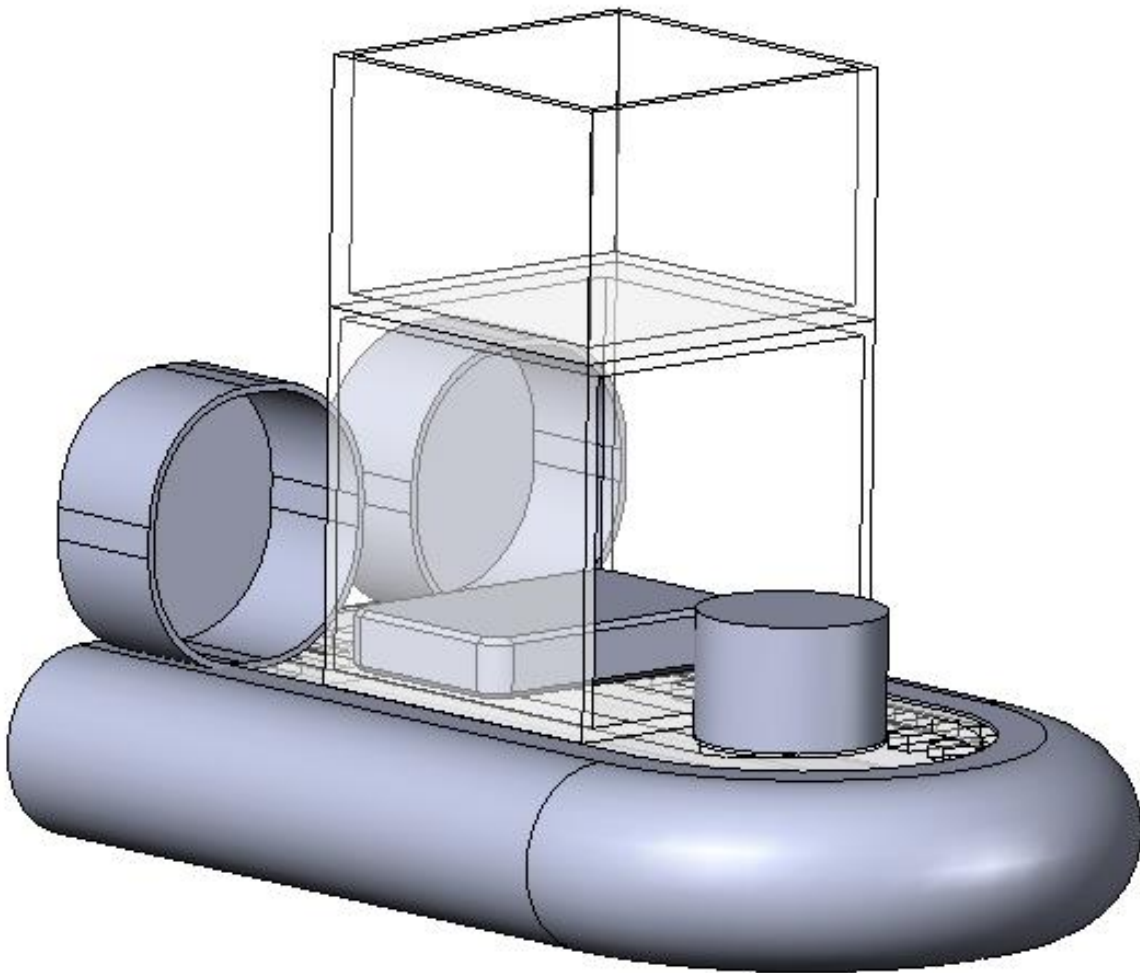


Figure 9: Fully assembled Prototype Concept

After concept selection, Boeing-1 found that concept H was the highest scoring concept. In order to select the best concept for design, the team decided to combine some of the stronger features of other concepts and remove some of the more complex features of the H concept. Thus, the chosen concept for design is a three fan hovercraft that uses a single fan for lift and two rear-mounted variable pitched propellers for thrust. The thrust propellers change their differential thrust to steer the craft as well as propel it forward or reverse. The bag skirt that supports the craft was chosen to enhance the simplicity and manufacturability to enable quick product turnaround.

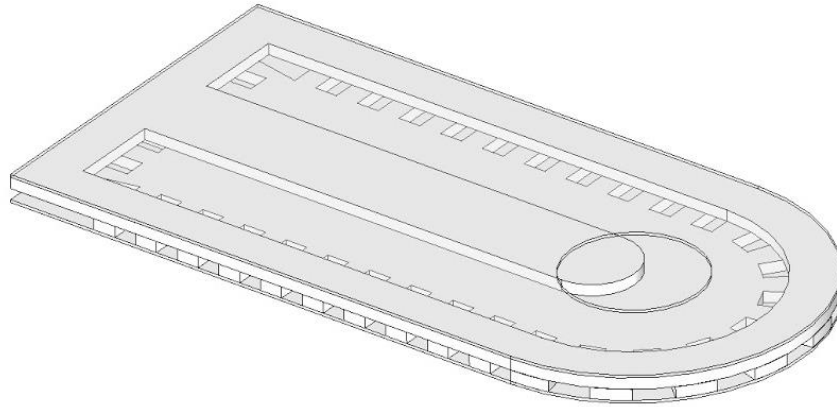


Figure 10: Integrated Hull/Plenum

The hull is one of the most unique features of this design. The hull will be comprised of four layers. The outer two layers are the structural layers that form the top and bottom of the craft's base. The middle two layers are designed to act as the plenum and air channels, guiding the air from the lift fan to the hull perimeter to inflate the skirt. This combined plenum design will reduce weight and allows for efficient use of space and materials.

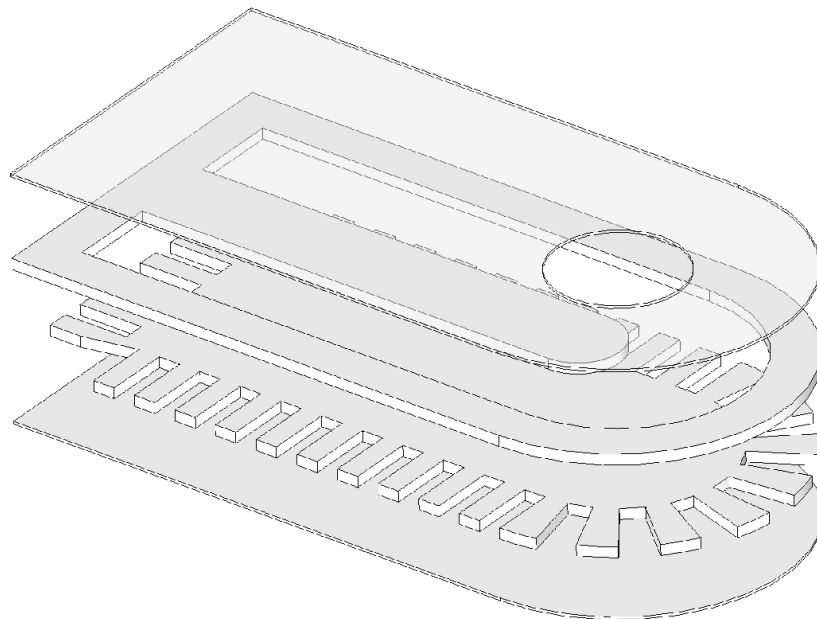


Figure 11: Hull/Plenum Exploded View

Fully assembled, the hull will be less than an inch thick and will provide adequate space for the lift fan and electronics on the hull surface. Although there are cavities internally, there is ample structure due to a beam that runs down the center of the second layer. Structural fasteners will be placed at points where all four layers have material. These solid points will also be useful for mounting structures on the base of the hovercraft.

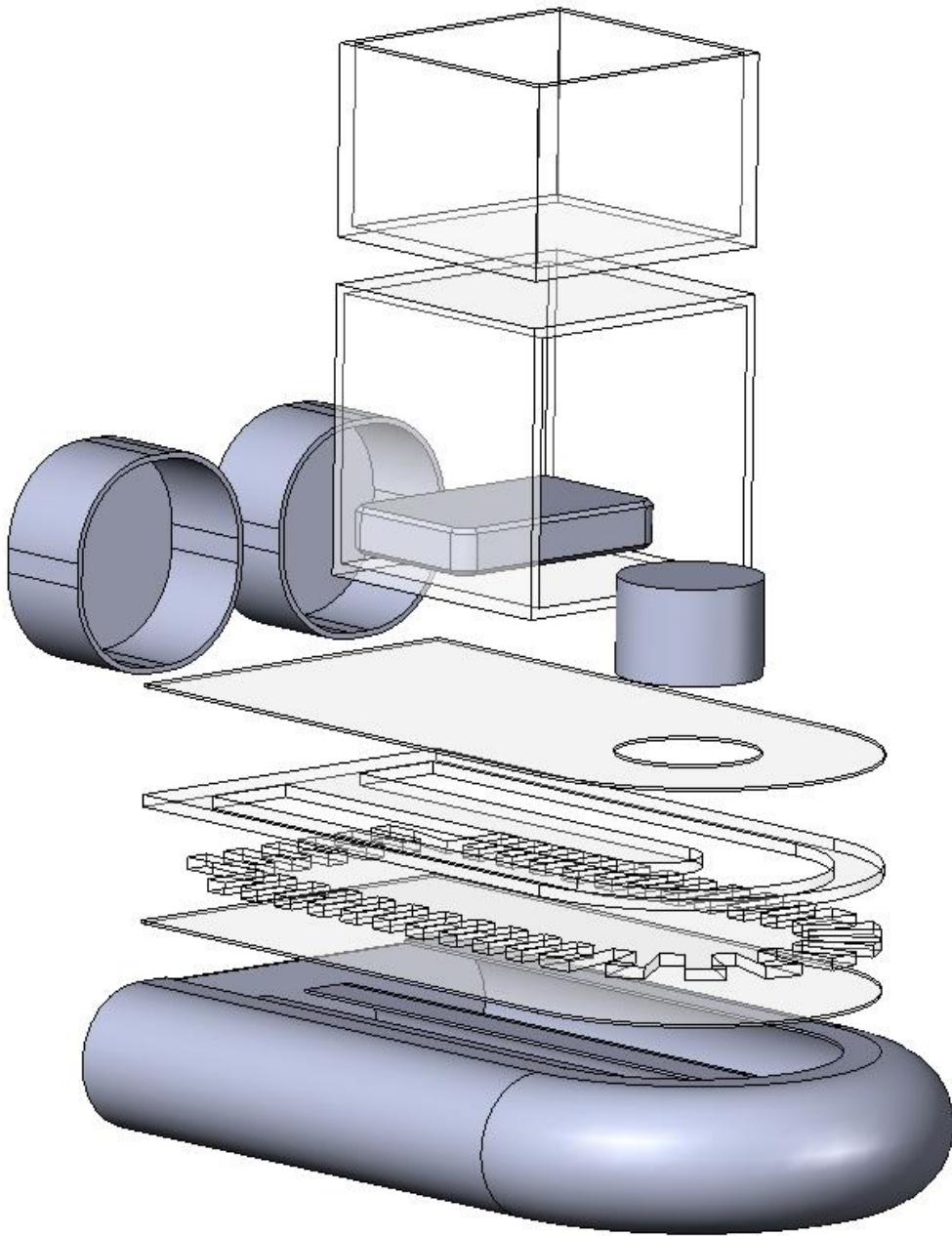


Figure 12: Full Assembly Exploded View

The design takes a simplistic approach to carrying the payload in order to simplify the design. The design utilizes a clear carrying container that is exactly the needed size in order to carry the payload.

7.0 Special Topics

7.1 Preliminary Economic Analyses - Budget and Vendor Information

Table 17: Preliminary BOM and Budget

Qty	Component	Web Link	Cost	Weight (g)
1	Propeller/motor	VPP / Motor	85.99	59.5
		Better Price	49.99	
1	Lift Fan	Lift Fan	38.5	88
1	Payload Bin	Payload Bin	8.99	522
1	Bulk Foam sheet	Foam	12.76	0
1	Polycarbonate Sheet	Polycarbonate	14.74	245
2	Li-po battery	Battery specs	0	694
1	MyRio 1900	MyRio Spec Sheet	0	193
		Nylon Skirt		
1	RipStop Nylon	Material	5.1	12.9
1	Silicone Spray	Silicone Spray	12.58	0
1	Payload	N/A	0	453.6
Total Weight				2268
Total Weight (lbs)				5.00
Total Cost				228.65

7.2 Project Management

Timely project management is a crucial tool that will aid in any design process. Boeing-1 used a Gantt chart as seen in Appendix B to manage deadlines and schedule deliverable dates. A deliverables agreement was drawn up in conjunction with the Learning Factory to document the date by which certain project deliverables were to be finished. With the teams' combined management and technical skills, the project deliverables will be satisfactorily accomplished by the required deadlines.

7.3 Risk Plan and Safety

The duration of the design project will likely have many plan changes and setbacks that could pose a potential risk to the successful completion of the hovercraft. The table below identifies, ranks, and addresses solutions to potential risks during the design process. Most customer relation risks are minimized by scheduling weekly customer meetings while risks internal to the group are minimized through careful planning to include deadline cushions. The high risks indicated below are shown in red and are likely to directly prevent the hovercraft from functioning while the moderate and low risks highlighted in yellow and red, respectively, could cause a setback of the schedule but would minimally affect the hovercraft design.

Table 18: Risk Management

Risk	Level	Actions to Minimize	Fall Back Strategy
Initial bag skirt concept prevents hovercraft from clearing 0.5” obstacles.	H	By designing the hull in such a way that many skirt styles are compatible allows for backup plans to remain possible after hovercraft construction begins. By fastening the skirt to the hull by non-permanent method, the skirt remains removable throughout the hovercraft build.	If the bag skirt does not allow for adequate floatation over obstacles and on uneven ground, either a finger skirt or a bag-and-finger skirt will be implemented as it allows for better floatation over obstacles but compromises stability.
Delays due to high lead time for supplier parts	H	Each supplier part will be ordered on the same week in which the decision was made to use that part in the final or prototype design.	If a supplier part lead time is too long to remain on schedule either a new supplier will be contacted the design will be modified to accommodate a different part.
Change of customer needs or specifications	M	By conducting weekly video meetings with the customer any change in needs or specifications will be known on a weekly basis.	Budget and time will be allotted in the schedule to accommodate testing prior to the product deadline that can be used to modify the craft if needs or specifications do change.
Customer needs not met	M	Weekly meetings in which the design is discussed will prevent any needs from being overlooked.	Ways to alter the design will be discussed with the customer within 48 hours of discovering the unmet need.
Hovercraft stability does not meet design requirements	M	The payload container/pedestal will be designed so that, if needed, a self-adjusting ballast weight can be added to the existing design without rebuilding the existing hovercraft.	If the hovercraft cannot remain stable on a 20 degree side slope, a self-adjusting ballast weight design will be added to the hovercraft that minimally increases total vehicle weight.
Part function does not meet supplier estimates	L	Each supplier part will be tested for functionality as soon as received (ex: test fan cfm or prop thrust)	All receipts from purchases will be kept throughout design process so returns can be attempted if a supplier part does not meet specifications
A design member must leave the team	L	Two weekly out-of-class meetings and two in-class allow team members to keep others informed of any change in person four days a week.	If a team member must leave the group, then the team will discuss the former member’s tasks and suggest how to divide them among the remaining members.

7.4 Ethics Statement

Engineers have a responsibility to be honest and forthright about the products they create. Boeing-1 will hold themselves to high ethical standards during every phase of the project. This will include providing accurate product specifications and alerting the sponsor to all the possible failures in the design. The design will not infringe upon any patents or intellectual property.

7.5 Environmental Statement

Boeing-1 will take into account the impact their design will have on the environment. The team will do this by choosing to use recyclable or renewable materials whenever possible. The hovercraft will be powered by rechargeable batteries and will use electric motors for lift and propulsion so there will be no emissions while the hovercraft is running.

7.6 Communication and Coordination with Sponsor

Boeing-1 will attempt to meet with the sponsor at least once a week via a Google hangout, or other telepresence solution. The team plans to meet every Wednesday at 8:00pm starting on September 10th. Each meeting will be expected to last one hour. The team does not plan to meet with the sponsors on November 26, as the week of November 24 is Penn State's thanksgiving break.

8.0 Detailed Design

8.1 Manufacturing Process Plan

Manufacturability and costs were key concerns during the design process. Below is a detailed list of how each part will be manufactured and then how these parts will be assembled into the final hovercraft.

8.1.1 Manufacturing Process Plan

Table 19. Manufacturing Process Plan

Assembly Name	Sub Part	Material Type	Raw Stock	Operations
Base	Acrylic Top Sheet	Acrylic Sheet	20" x 32" x .093" sheet	Laser Cut
	Acrylic Bottom Sheet	Acrylic Sheet	20" x 32" x .093" sheet	Laser Cut
	Foam Upper Layer	Foam Sheet	20" x 32" x .093" sheet	Laser Cut
	Foam Bottom Layer	Foam Sheet	20" x 32" x .093" sheet	Laser Cut
Payload Containers	Full Acrylic box	Acrylic	Modified Purchase Part	Drill 4 size 4 holes in the bottom of the box Cut a slot in the bottom of the box on a mill beside the holes leaving .75" on each side of the box. Completely remove one face of the box on the mill Cut out holes for one of the battery on the mill
	4" Acrylic box	Acrylic	Modified Purchase Part	Cut 2 inches off the open portion of the box on a mill to make the dimensions 6" x 6" x 4"
Skirt	Bag Skirt - Back	Ripstop nylon	Fabric	Laser Cut
	Bag Skirt - left and right	Ripstop nylon	Fabric	Laser Cut
	Bag Skirt - top - center	Ripstop nylon	Fabric	Laser Cut
	Bag Skirt outer	Ripstop nylon	Fabric	Laser Cut
Gearbox	Jackshaft Bearing Fixture	Acrylic	20" x 32" x .093" sheet	Laser Cut
	Jackshaft Bearing Fixture Boom	Acrylic	20" x 32" x .093" sheet	Laser Cut
	Jackshaft Bearing Fixture Case	Acrylic	20" x 32" x .093" sheet	Laser Cut
	Jackshaft Bearing Fixture Rear	Acrylic	20" x 32" x .093" sheet	Laser Cut
Lift Fan	Lift Fan Mounting Piece Bottom	Acrylic	20" x 32" x .093" sheet	Laser Cut
	Lift Fan Mounting Piece Top	Acrylic	20" x 32" x .093" sheet	Laser Cut
Thrust	Thrust Shrouds	PVC	4.75" x 1'	Cut to length on band saw Drill 3 countersunk size 4 holes in the PVC
	Various Purchased Parts	NA	NA	Assemble using directions from the Align T-rex 250 manual

8.1.2 Summary of Assembly Process Plan

The multiple assemblies detailed above will be joined together to form the final prototype. We decided to use #4 zinc coated machine screws to join solid assemblies to the hovercraft base, and adhesive to join the four layers of the hovercraft base assembly. We selected to use #4 machine screws because they were sized properly for smaller assemblies, and strong enough to hold larger

assemblies in place reliably. These machine screws are also abundant and inexpensive, cutting down on manufacturing cost.

Table 20. Assembly Process Plan

Assemblies	Assembly Summary
Base Platform	Assembly of bottom base platform with skirt and then bolt the top base platform.
Skirt	All 7 pieces will be turned inside out and sewn together. The complete skirt is then glued in between the bottom acrylic piece, the bottom base and the top base, then it is fastened by placing the top section of acrylic and securing it with #4 machine screws.
Lift Fan to Base	The lift fan along with the acrylic pieces will be bolted to the base using (4) # 4 size machine screws, and two acrylic pieces whose function is both holding the motor in place and keeping users from getting hurt by the fan.
Thrust Assembly	The thrust assembly consists of a gear and motor support made out of successive acrylic pieces which as well as the T-rex parts for the variable pitch propeller fans. The thrust assembly will be mounted on the base using two cut acrylic pieces bolted into the base using (4) # 4 machine screws.
Payload Containers	The payload containers will be bolted to the base using (4) # 4 machine screws
Batteries and MyRio	The batteries and MyRio will be attached to the hovercraft using Velcro

8.1.3 Detailed Assembly Plan

Base Assembly:

1. Use adhesive to glue bottom part of the skirt to the bottom acrylic piece making sure that the skirt fabric goes inside the inner hole of the acrylic piece, bends over the edges and finally glues to the upper surface of the acrylic without obstructing the cuts meant for air flow.
2. Glue the assembly in step 1 to the bottom surface of the first foam piece of the base, making sure that both pieces align.
3. Glue the next foam section to the previous assembly following the procedure in section 2.
4. Glue the skirt to upper foam part of the base so making sure to glue $\frac{1}{2}$ of an inch of the skirt uniformly over the upper surface of the foam. When this step is finished, the first section of the assembly has been completed, see Figure 13 for clarification.

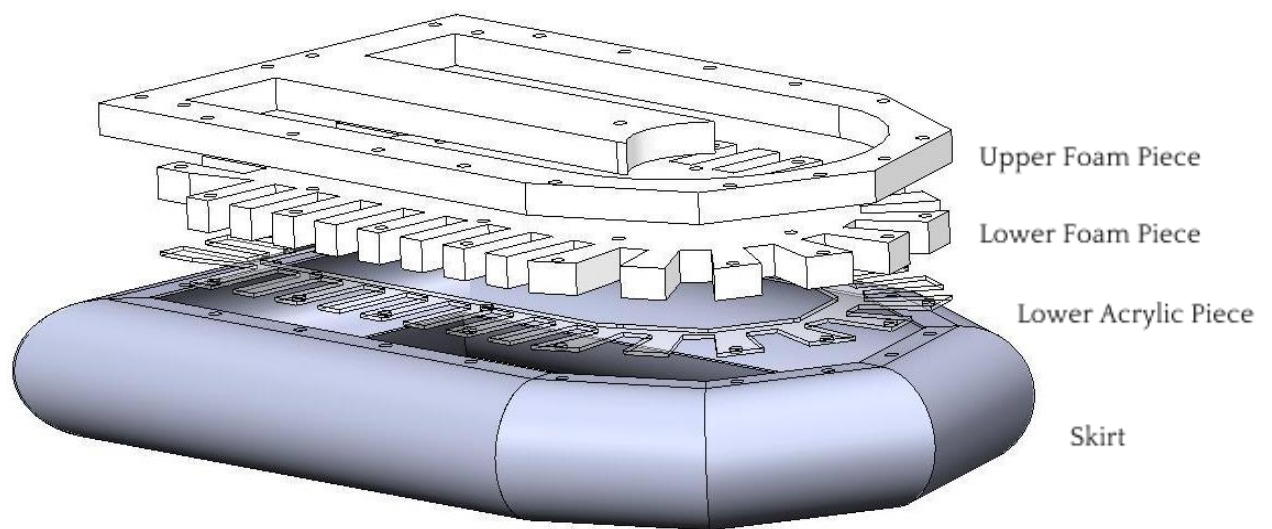


Figure 13: Skirt and Base Assembly

Thrust Assembly

1. Fasten the gear to the motor using adhesive.
2. The gearbox consists of successive acrylic sheets that come together to hold the thrust assembly in place. The first four of these acrylic pieces will secure the motor and the gear (step 1) and the two belt pulleys in place.
3. Connect the boom support and boom to the fourth acrylic piece and secure it using the remaining pieces. See Figure 14 for acrylic piece alignment.

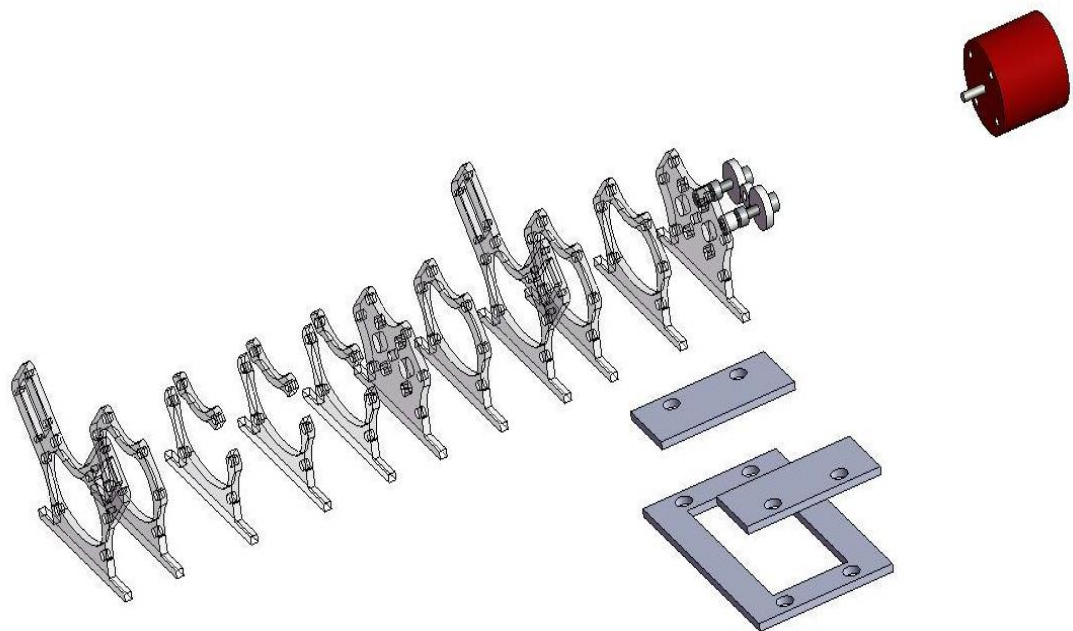


Figure 14: Gear Box Assembly

4. Assemble the variable pitch propellers and servos to the boom using the Align T-rex instruction manuals and support parts.
5. Place the gear box support bracket on the upper acrylic piece of the base,
6. Place the gearbox assembly (steps 1-3) inside the bracket.
7. Use the securing plates and #4 machine screws to fasten the assembly to the acrylic piece.
8. Screw the 4 aluminum standoffs to the base.
9. Secure the thrust shrouds to the aluminum standoffs using size 4 machine screws
10. Attach another aluminum standoff between the shrouds to avoid lateral displacement.

Payload Container

1. Take the upper base piece of acrylic with the thrust assembly and place the payload support on top of it so that the holes in the support are aligned with those in the acrylic piece.
2. Use # 4 machine screws to secure the payload support to the base.
3. Attach the payload support to the payload container with adhesive as shown in Figure 15.

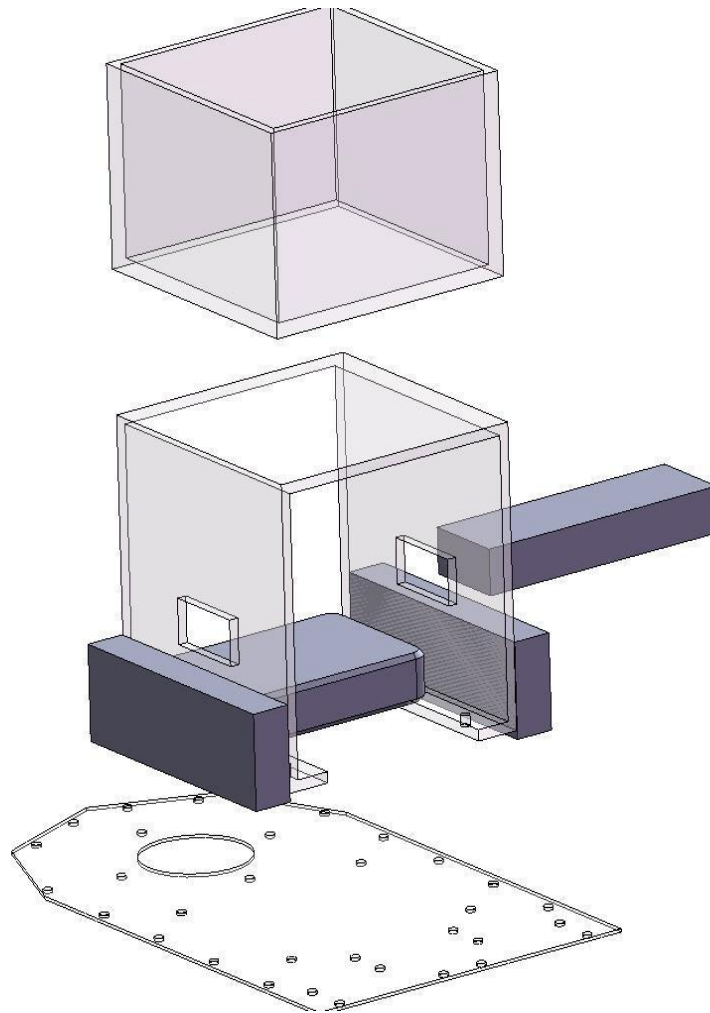


Figure 15: Payload, Arduino and Battery Assembly

Lift Fan Assembly

1. Secure the bottom lift fan circular bracket to the top base acrylic piece with the payload container and thrust assemblies already attached.
2. Place the lift fan within the securing bracket.
3. Place the upper securing bracket over lift fan.
4. Secure parts using #4 machine screw.

Hover Craft Assembly

1. Take the fully assembled upper acrylic part secure it to the lower base assembly with the skirt using the specified screws.
2. Paste Velcro in respective places to hold the batteries, MyRio and receiver in place.
3. Place components and make connections.
4. Insulate cables.

8.2 Analysis

In order to adequately prepare for prototype testing, some preliminary calculations must be completed to verify the structural integrity of certain components that will experience significant stresses during operation. During hovercraft operation, significant stresses will be experienced by the tail booms, which support the thrust propellers. These components are of concern due to their thin walled construction and aluminum material. The analysis below details how Boeing-1 calculated the maximum thrust required to propel the craft. Maximum thrust will be required when propelling the vehicle up a 20 degree slope. The thrust must overcome the component of the weight that is directed down the slope as well as the friction due to the normal force.

The Matlab code in Figures 17 and 18 was developed to calculate the thrust required for a hovercraft weight of 6 lbs, which is the closest estimate to actual weight. It also calculates the tensile stress and deflection experienced by the boom during that maximum thrust event. Figure 16 shows the free body diagram that was used to develop the force balance.

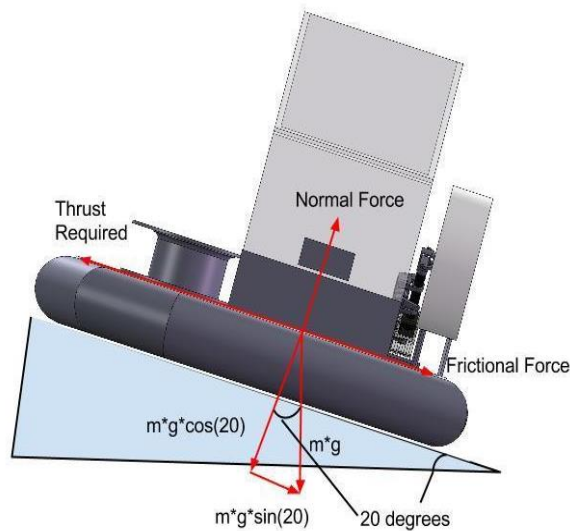


Figure 16: Free Body Diagram for Force Balance

```

% Calculation of maximum thrust

% Given
W = 6; % Hovercraft weight range (lbs)
a = 20; % Maximum Incline (degrees)
mu = 0.0367; % Coefficient of kinetic friction (found experimentally)
l = 74.48/25.4; % Length of moment arm for one boom(in)
Do = 8.5/25.4; % Outer diameter of boom (in)
Di = 7.9/25.4; % Inner diameter of boom (in)
FS = 2; % Factor of safety
C = Do/2; % Outer radius of cylinder (in)

% Maximum Thrust Calculations
TotalThrust = W*sind(a)+mu*W*cosd(mu); % (lbs)
ThrustPerProp = TotalThrust./2; % (lbs)
MaxThrustPerProp = FS*ThrustPerProp; % (lbs)

% Bending Stress Calculations
Z = (0.78*((Do/2)^4-(Di/2)^4))/(Do/2); % Section Modulus (in^3)
I = 0.78*((Do/2)^4-(Di/2)^4); % Moment of inertia (in^4)
E = 10007603.9; % Modulus of elasticity for T-6061 Aluminum (lb/in^2)
Stress = (W*l)/Z; % Maximum bending stress in boom (lb/in^2)
Deflection = (W*l^3)/(3*E*I); % Boom deflection (in)

```

Figure 17: Matlab Code Thrust/Deformation Calculations

```

TotalThrust =

    2.2723

ThrustPerProp =

    1.1362

MaxThrustPerProp =

    2.2723

Stress =

    1.8969e+04

Deflection =

    0.0325

```

Figure 18: Results of Thrust/Deformation Calculations

Following the analysis, Boeing-1 could conclude that the booms were robust enough to withstand the maximum thrust as the max stress experienced with a factor of safety of 2 was approximately 18,000 psi and the yield stress of T-6061 aluminum is 40,000 psi. Boom deformation was calculated to be approximately 0.0325 inches, or 0.83 mm. Figure 19 show a subsequent model that was developed to conduct a rudimentary finite element analysis, or FEA, to verify the manual calculations. The FEA models proved the accuracy of the hand calculations to be on the correct order of magnitude and less than the yield stress of aluminum. The FEA models show the highest stresses in red. In Figure 19 the highest stress, with no factor of safety, is shown to be in the 5,000 psi range. After the theoretical analysis, Boeing-1 continued testing to further understand the behavior of the alpha prototype hovercraft.

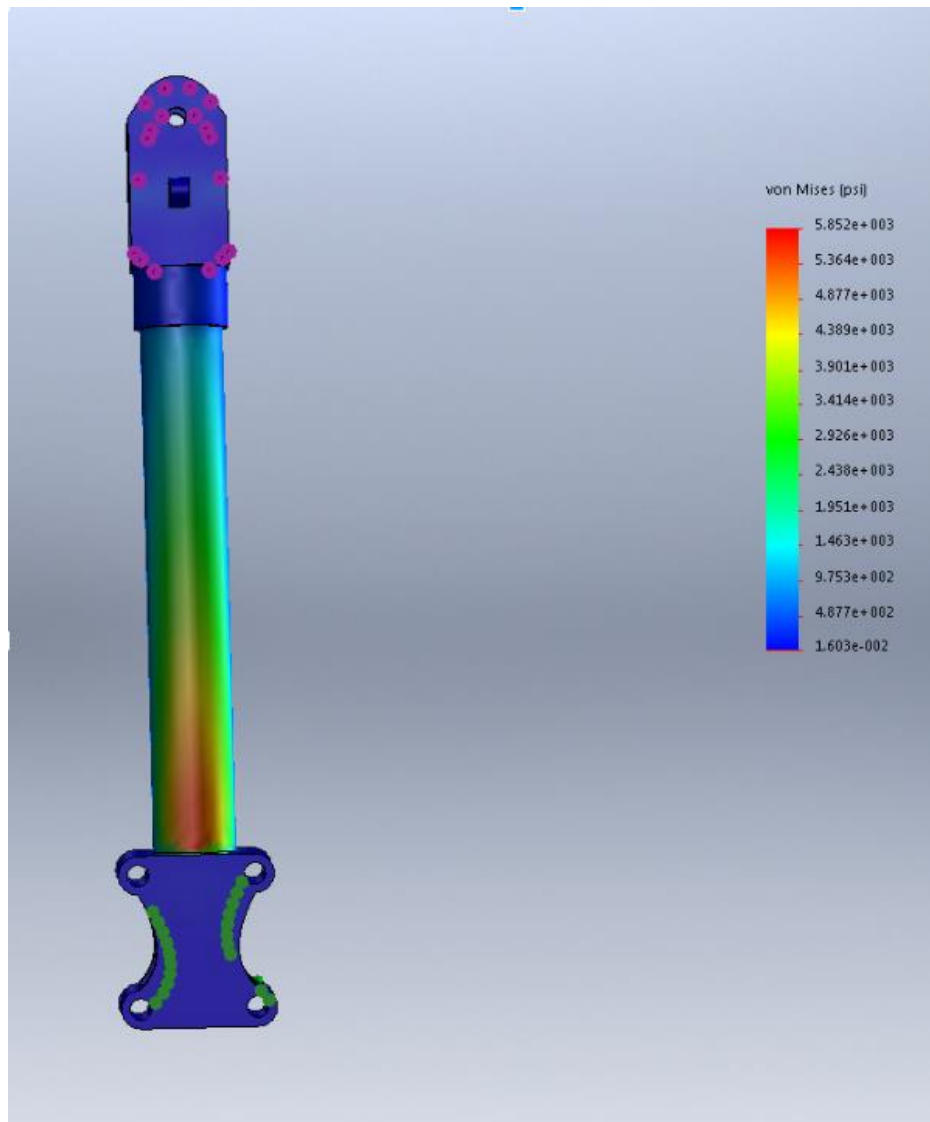


Figure 19: SolidWorks FEA Model

8.3 Material Selection Process

The major material choices for the prototype were in the skirt, hull, and rear thrust fan assemblies. Our goal in choosing materials was to select materials that could be easily sourced and cost effective, while still allowing us to reach target specifications for the vehicle.

Skirt

For the skirt, we chose to use lightweight and strong nylon fabric. This type of fabric is able to be cut in a laser cutter, allowing us to cut the skirt pattern more precisely. Nylon fabric also allows us to sew strong seams to hold the skirt together while preventing the skirt from fraying around the edges. Nylon fabric is readily available in inexpensive bulk from many craft and fabric stores.

Outer Hull

The top and bottom of the hovercraft hull will be made out of 0.093 inch acrylic plastic. This material was chosen because of a need for a hard outer shell to protect the soft inner foam hull from damage. The harder acrylic also provides rigid mounting points for components on the top of the hull such as the lift fan and the thrust fan assembly. Acrylic also provides a significant advantage over other plastics in manufacturing. While many plastics produce toxic fumes in a laser cutter, we are able to laser cut the acrylic without these safety concerns.

One disadvantage of using acrylic is in cutting screw holes with a laser cutter. We found that even though we could use CAD to design an assembly with a specific screw size, tolerances in the laser cutting process produced larger than intended holes in the resulting assembly. This property is especially concerning as we attempt to cut bearing holes in the acrylic for the rear thrust fan assembly. We have run some preliminary tests, and have found a relationship between CAD hole diameter and actual finished diameter. These results should help us correct for the tolerances in the laser cutting manufacturing.

Inner Hull

The primary function for the inner hull is to route airflow from the lift fan to the skirt. This function requires that the passages in the inner hull be large enough, as well as rigid enough to withstand high airflow. Although the inner hull should have a large volume, it should not unnecessarily encumber the hovercraft. Because of these requirements, we chose to use two layers of ½ inch paper-backed Styrofoam craft board. This material is easily laser cut, and allows us to obtain the internal channel width necessary to support high airflow.

One disadvantage to the Styrofoam material is its lack of strength and general malleability. The Styrofoam is easily punctured and torn by impacts, and thus lacks the rigidity to be sufficient for the entire hull. To make up for these deficiencies, we chose to layer acrylic plastic on the top and the bottom of two layers of Styrofoam. The acrylic protects and supports the Styrofoam, and gives the hull all of the qualities we need to meet our target specifications.

Thrust Assembly

The rear thrust assembly poses a unique set of challenges, as it requires us to adapt a set of off the shelf commercial R/C helicopter parts in an alternate drive set up as well as firmly affix the drive assembly to the hovercraft. Because of the numerous requirements of the assembly, we developed a CAD gearbox drive made of sandwiched layers of acrylic plastic. Using acrylic allows us to easily construct new revisions quickly and cheaply with the laser cutter. The acrylic is also much lighter than a machinable material such as aluminum, reducing weight in the rear of the craft.

8.4 Component Selection Process

In selecting components for the prototype construction, we had to keep in mind cost effectiveness and time-effectiveness while still producing a prototype that can successfully validate our design.

Skirt

After our pre-alpha prototype with a hand-cut nylon skirt, we decided that it would be beneficial to ensure that it would be beneficial to construct a laser-cut nylon skirt for the final prototype. We anticipate that cutting the nylon with the laser cutter will be a low risk way of ensuring an effective skirt production process. The laser cutter will also be able to produce an even more accurate resulting skirt, giving us a more representative final prototype.

Hull

The hull will be constructed using two laser-cut pieces of paper-backed Styrofoam craft board sandwiched between two sheets of laser cut acrylic plastic. We will fasten the four layers together with #4 machine screws at various locations around the hull perimeter. In order to accommodate unforeseen additions of mounted components, we will be manufacturing the hull with extra mounting holes on its perimeter.

Thrust Assembly

The components for the thrust assembly were chosen by researching different models of R/C helicopters. We found that one particular model, the Align T-Rex 250, used a variable pitch tail rotor with a 4.25 inch diameter. This diameter was ideal for mounting two rotors on the rear of the hovercraft for propulsion. In assembling the entire rear thrust component, we needed to build an adapting assembly to drive the propellers with belts from geared shafts. For this assembly, we decided to use as many of the prefabricated R/C helicopter components as possible. This allowed the work on the adapting assembly to be as minimal as possible, with most of the complexity involving the drastic shortening of the drive belt to the rotor. The entire rear thrust component is manufactured with acrylic plastic and fastened to the hovercraft base with an acrylic mounting bracket.

Protecting the thrust fans is also necessary for both user and machine safety. We decided to use a small slice of PVC tubing to shroud each of the rotors. The PVC slices will be mounted to the hovercraft hull using 1 ¼ inch standoff posts, and will be guarded by wire mesh.

8.5 CAD Drawings

Solidworks was used to create 3D models of the manufactured parts as well as the purchased parts that will make up the RPH. The four Figures below show the CAD model of the fully assembled hovercraft. Drawings for the manufactured parts can be found in Appendix F. The mass that will be added during the obstacle course test were not added to the assembly because the size and shape of these objects are not yet known.

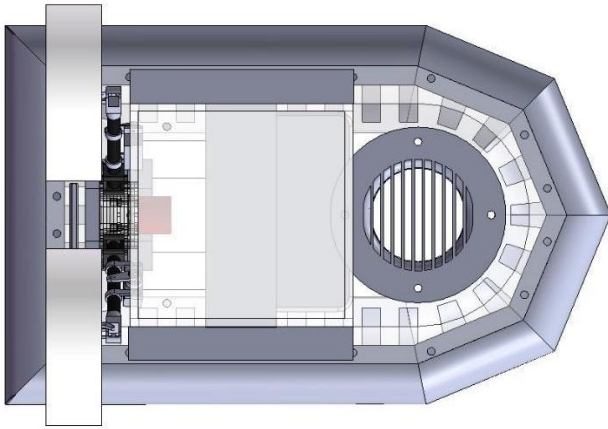


Figure 20: RPH Top View

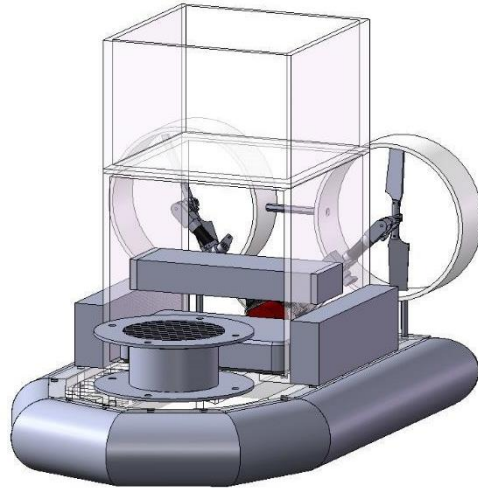


Figure 21: RPH Isometric

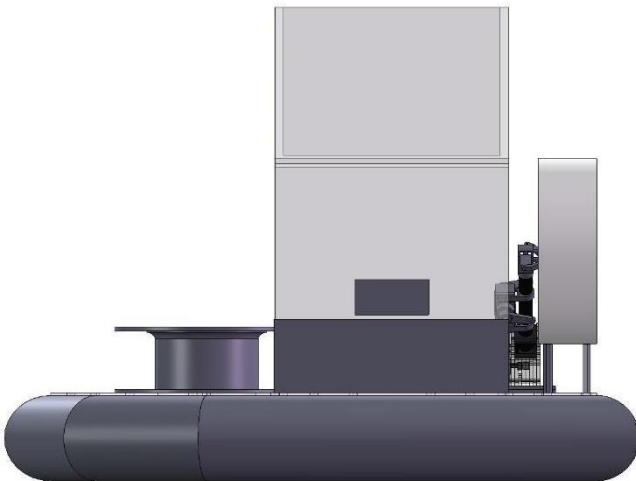


Figure 22: RPH Right View

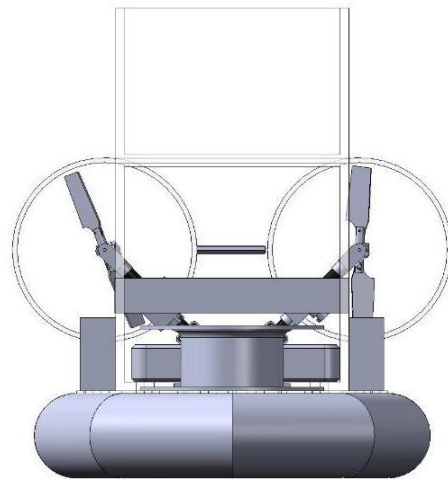


Figure 23: RPH Front

8.6 PID Heading Control

8.6.1 Mechanical Design Modifications

After deciding that the radio controls were to be removed in order to independently develop the PID heading controls for the hovercraft, it was also noticed that some of the components were no longer needed. The changes made are outlined below:

- Changed Arduino UNO microcontroller for a RedBot microcontroller.
- Removed payload Upper box.
- Installed a master switch to turn on and off the power for the whole hovercraft.
- The DC Motor that failed during testing was replaced with a spare

The changes listed previously are shown below in Figures 24-27.



Figure 24: PID Front View

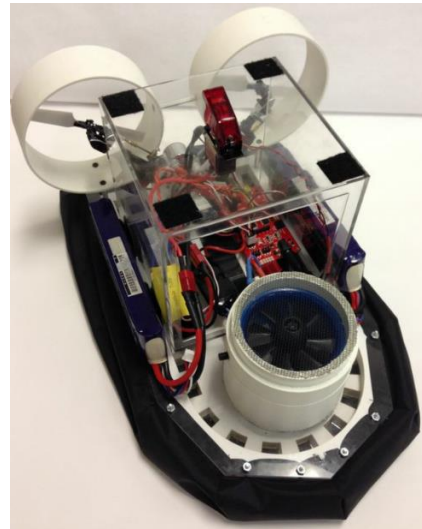


Figure 25: PID Isometric View

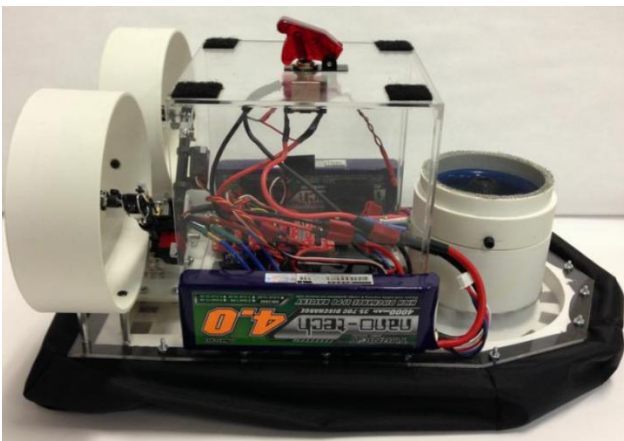


Figure 26: PID Side View

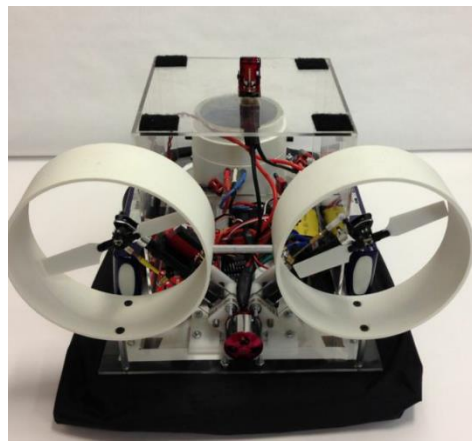


Figure 27: PID Back View

8.6.2 PID Controls

The PID control that was implemented for the heading stabilization of the hovercraft used feedback from a gyro/accelerometer. Although only the z-axis gyro signal was used for the control, it provided very valuable rotational rate information that was necessary for maintaining heading.

The hovercraft's PID control differs from typical motor position control in two ways. The first is that instead of a voltage value to command clockwise or counterclockwise rotation, the hovercraft rotation is controlled by the position difference between two servos. The servos on the hovercraft pitch the thrust propeller blades to produce either forward or reverse thrust. By pitching the blades opposite each other, a torque can be applied to the hovercraft and cause the body to rotate.

The second difference is the feedback data. The gyro provides a rotational rate instead of a position that would typically be received from an electric motor's encoder. This means that the proportional control actually acts like a derivative when considering the hovercraft's heading angle. If only proportional control were implemented, the hovercraft would rotate and, after much oscillation, be brought to a stop at a random heading. Derivative control was required to serve as a damper. Since the gyro provides rate data, the derivative controls the rotational acceleration of the hovercraft. As the hovercraft reacts to the proportional control and begins slowing down, the derivative control further decreases the torque applied by the propellers in an attempt to prevent overshoot. With only PD control, however, the hovercraft would stop rotating at an undesired heading. To produce a signal that will bring the hovercraft to its original heading, integral control must be applied. Integrating the rate of rotation allows for the angular position of the hovercraft to be tracked. As the hovercraft rotates, the integral control sums its motion in one direction and applies the appropriate corrective torque. As the body reacts to the torque and rotates back, the sum begins to decrease until the original heading is reached. PID control allows for the hovercraft to be disturbed and rotate back to its original heading with little overshoot.

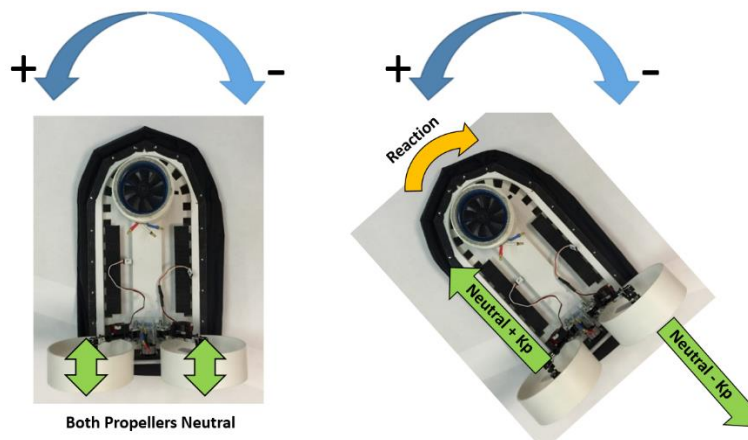


Figure 28: Proportional Gain

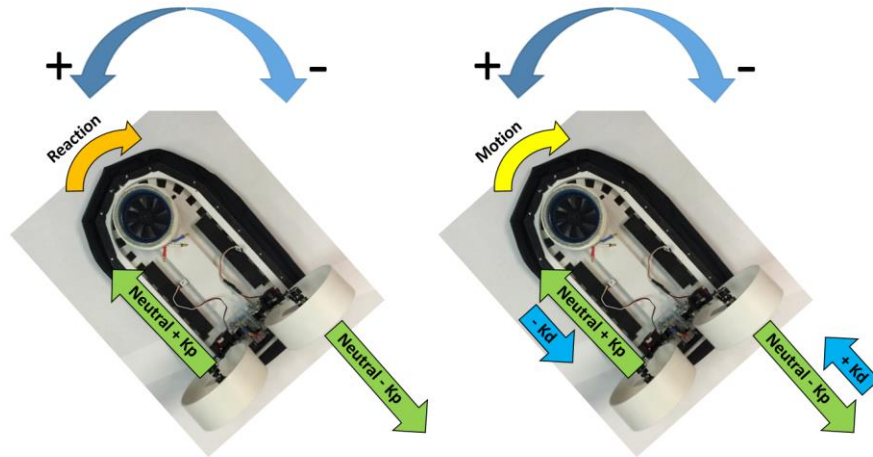


Figure 19: Derivative Gain

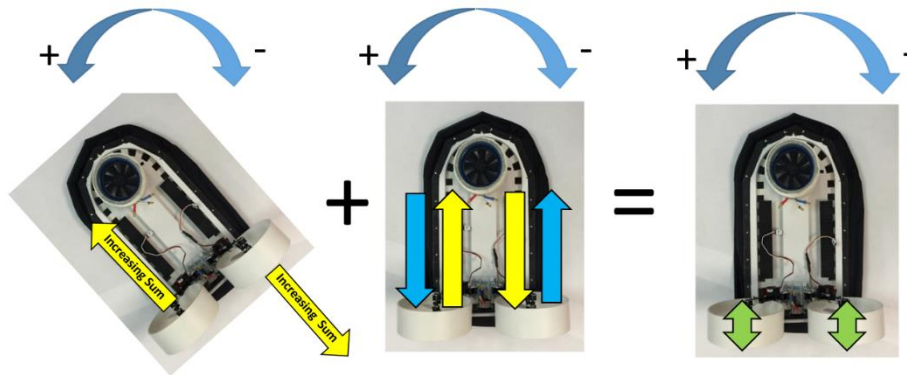


Figure 30: Integral Gain

8.7 Testing

8.7.1 Testing Procedure

In order to ensure that the hovercraft performs to specifications, we needed to devise and execute tests to validate the design. The main goal of these tests were to verify that the design could operate effectively in a variety of circumstances and under different loads. These tests also allowed us to quantify the performance of the design and to identify improvements as we adjusted and refined the design.

Battery Current

Before performing tests on battery life of the prototype, we performed some tests to identify key characteristics of the battery over the course of a single discharge. We measured the current drawn from the battery as a function of time at two different throttle percentages. The test was performed with a craft weight of 6 pounds, which most accurately represented the designed hovercraft weight. An inductive current probe with 5-400 amp range was used to measure the current drawn from the battery. Two trials were conducted, one with the throttle held at 36% and the second with

the throttle held at 44%. Current measurements were taken every 30 seconds for a duration of 15 minutes. As long as the battery can power the lift fan for 15 minutes it will meet the project requirements.

Electronic Speed Controller Temperature

In order to ensure that the electronic speed controller (ESC) operates in a safe temperature range, we performed tests to measure the ESC's temperature over time. We measured the temperature of the ESC as a function of time at two different throttle percentages. Like the battery current test, this test was performed with a craft weight of 6 pounds, which most accurately represented the designed hovercraft weight. An infrared temperature sensor was used to periodically take temperature readings from the ESC during operation. Two trials were conducted, one with the throttle held at 36% and the second with the throttle held at 44%. Temperature measurements were taken every 30 seconds for a duration of 15 minutes. At the beginning of the tests, the ambient temperature was 60° F. By the end of the test the ambient temperature had dropped to 54° F. The ambient temperature is important to know in order to understand why the steady state temperature of the ESC may decrease.

Current Draw Tests

One of our main concerns with the design was that the hovercraft would not have enough battery capacity to effectively operate for 15 minutes in competition. With our pre-alpha prototype, we validated that the skirt and base design were capable of stably hovering up to 7 pounds of load, but we needed to quantify whether or not we could actually perform for long enough under this sort of loading.

The most significant current draw on the battery comes from the main lift fan of the hovercraft. Since this lift fan runs constantly throughout the operation of the vehicle, it is constantly draining the battery. With the assumption that current draw is constant throughout the operation of the vehicle, we can approximate the maximum running time of the hovercraft by examining the current draw of the lift motor in different conditions. We identified three variables that could significantly affect the current draw on the lift motor: operating surface, throttle position, and payload weight.

Given an operating condition (surface, throttle, weight), we performed a current draw test by executing the following procedure. To perform the test, the unpowered prototype was placed on the operating surface with an inductive current probe attached to read the output current of the battery. In these trials, the operating surface was either 1½ inch AstroTurf or ⅛ inch AstroTurf. The appropriate weight was then placed on the the prototype. These trials were performed starting with 2 pounds of payload and incremented by one pound up to 7 pounds of payload. Once the payload was secured, the throttle was ramped up to the tested percentage. Our trials tested the throttle at 20%, 28%, 36% and 44%. Once the craft reached a steady state of operation, the reading on current probe was recorded. Below in Figure 31 is a picture of the test rig for the current draw tests.

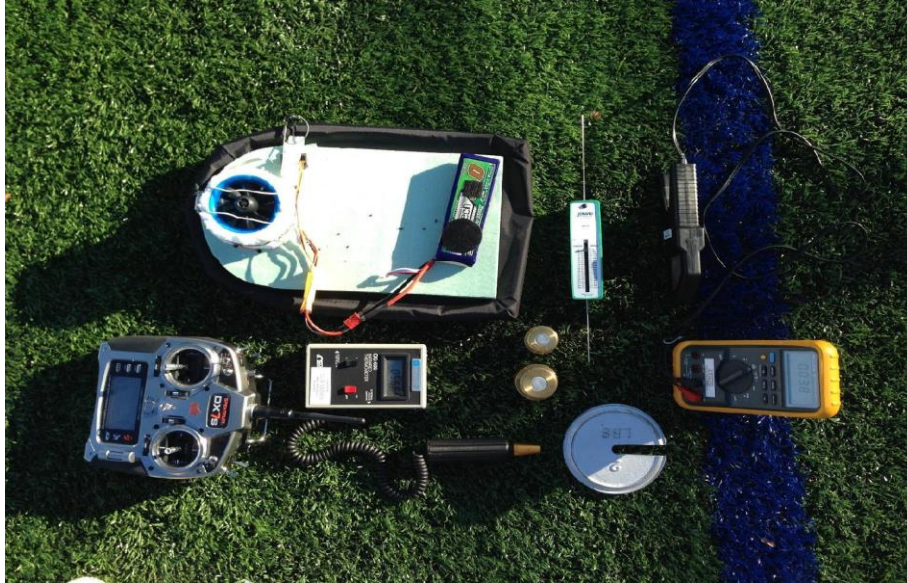


Figure 31: Test Rig Beta Prototype

Friction Tests

The hovercraft needs to be able to move across the operating surface. Since the performance of the craft is judged by how many circuits it completes in 15 minutes, lower friction will improve speed during operation and directly affect the overall performance of the vehicle. In order to measure the kinetic frictional force on the hovercraft, a spring scale with 1 kg maximum and 20 gram resolution was used to measure the force needed to drag the hovercraft across the operating surface. Throttle was varied from 20% to 44% during the trials, and kinetic frictional force was measured at each 8% throttle increment. Two trials were conducted on the $\frac{1}{8}$ inch AstroTurf, one with 24 cm² of hole area in the skirt, and one with 36 cm² of hole area in the skirt. The weights used to load the hovercraft during the kinetic friction tests are shown in Figure 32.



Figure 32: Weights for Friction Testing

8.7.2 Testing PID Gains

In order to determine the correct values for the proportional, derivative and integrative gains, the team did many iterations to calibrate each of the gains and determine which values provided the most satisfactory performance of the PID heading control. The derivative gain was most important as the hovercraft has very little friction with the ground causing it to easily overshoot. This meant that the derivative gain (damping) must be higher than the proportional and integral gains.

8.7.3 Testing Results Analysis

As described in section 8.5 above, in order to validate the hovercraft design, multiple tests were conducted on specific system parameters. The purpose of the alpha prototype tests was to determine the relationships between:

- ESC (Electronic Speed Controller) Temperature vs. Time
- EDF (Electric Ducted Fan) Current vs. Time
- EDF Current vs. ESC Temperature
- EDF Current vs. Throttle Position
- Kinetic Friction Force vs. Throttle Position

Each of these relationships indicate an important detail about the hovercraft design that can be studied to improve the overall design.

Figures 33 and 34 show how current and temperature change during a 15 minute operating cycle of the alpha prototype hovercraft for two different throttle positions. Individually, these plots tell us that the current drawn by the EDF falls to an equilibrium after approximately 3 minutes. The temperature responds in almost the exact same manner; however, the temperature of the lift fan ESC rises to an equilibrium in the same amount of time. This relationship was expected by the team members. More importantly, the test indicated that the 4000 mAh battery was sufficient to provide power to the lift fan for the entire 15 minute operating cycle. Originally the design team predicted that three batteries would be required for hovercraft power. After conducting the current tests Boeing-1 became confident that the hovercraft will only require two batteries for 15 minutes of operation; one battery dedicated to lift and one dedicated to thrust.

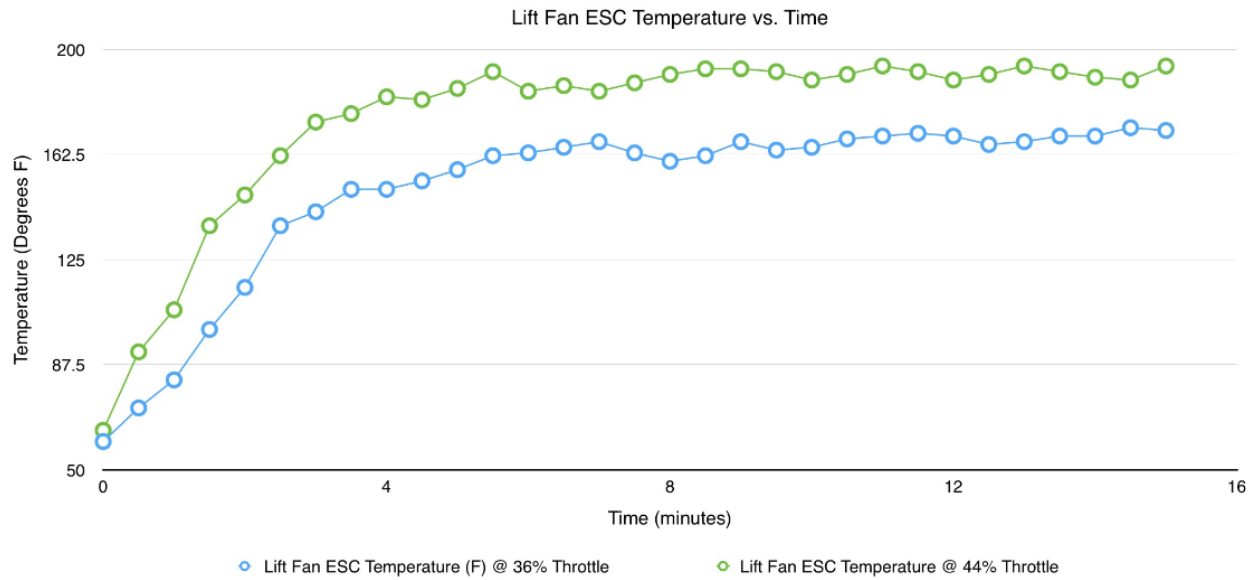


Figure 33: Temperature vs. Time

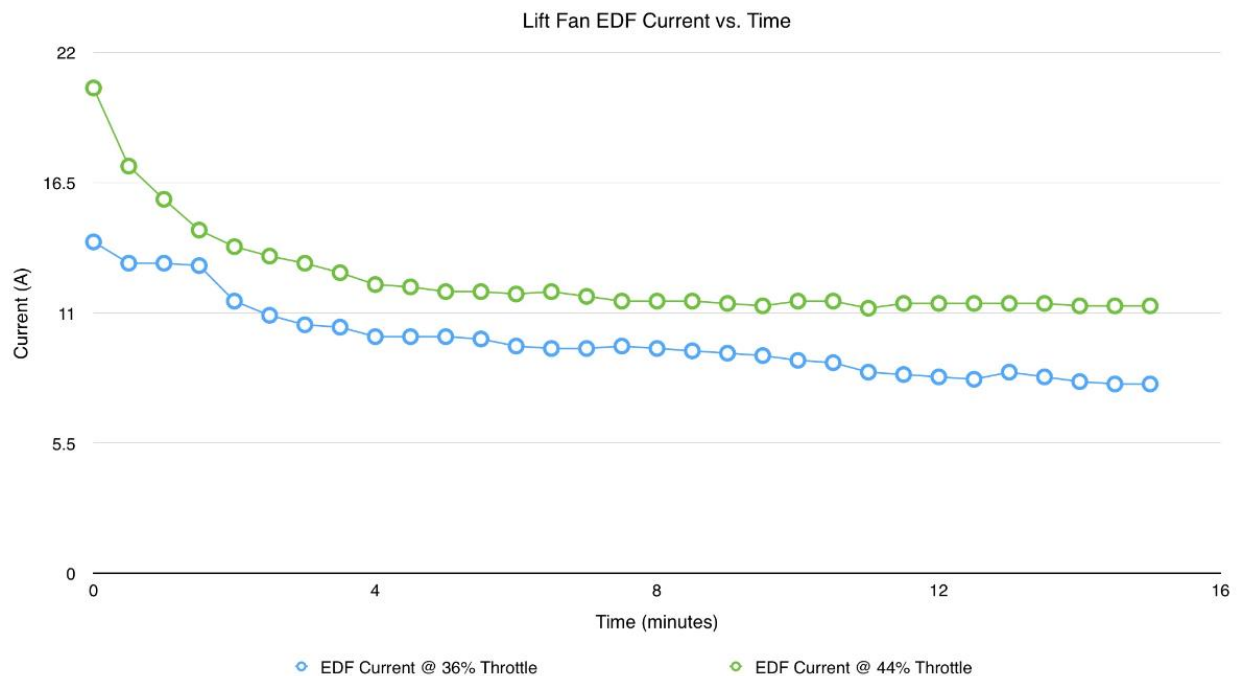


Figure 34: Battery Current

Another conclusion that can be drawn from Figures 33 and 34 is the relationship between current and temperature. Although both lift fan current and ESC temperature are clear functions of time, they can be related to each other. Figures 35 and 36 show how the current changes as a function of ESC temperature for throttle positions of 36% and 44%, respectively. Regardless of whether this relationship is direct it did indicate to Boeing-1 that managing the ESC temperature is very important for consistent current.

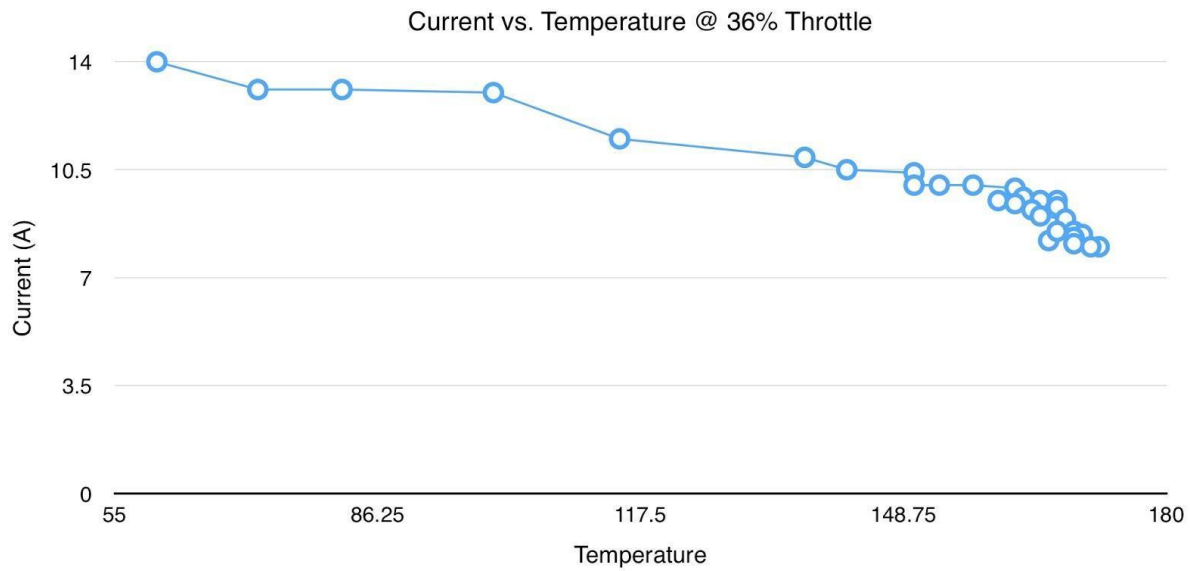


Figure 35: Current vs. Temperature at 36% Throttle

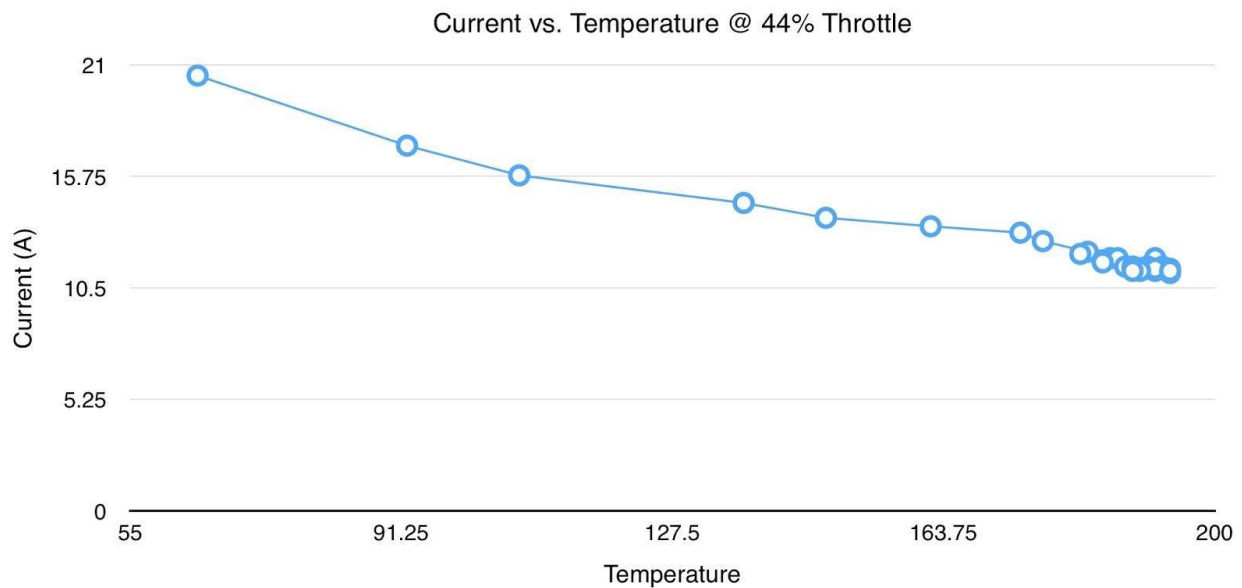


Figure 36: Current vs Temperature at 44% Throttle

Figures 37, 38, and 39 show the current drawn by the lift fan as a function of throttle position for different operating conditions. The operating conditions that change between figures are the length of the Astroturf and the total hole area in the underside of the skirt. All three figures show a positive linear trend for the current as a function of throttle position. Multiple trials with different loads were conducted for each operating condition. The results show that the current drawn by the lift fan is only dependent on throttle position. This result was contrary to the team's initial assumption that the current drawn by the fan was directly related to the hovercraft weight.

It can be seen in Figures 30 through 32, that the current drawn by the lift fan consistently decreased as the load was increased. This phenomena was actually due to the order in which the test was completed. Lower hovercraft weights were tested first and then weight was added. As shown in Figure 27, the current will decrease as a function of operating time. This means that as the test proceeded, the current drawn by the lift fan decrease slightly causing the downward shift in data points for each throttle position in Figures 30 through 32.

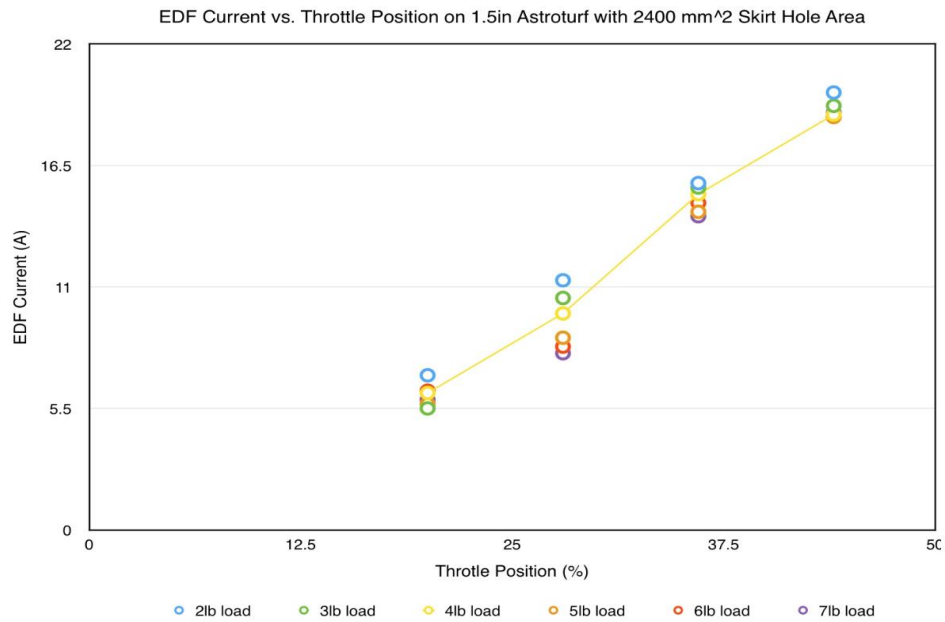


Figure 37: EDF Current vs Throttle 1.5 in, 2400mm²

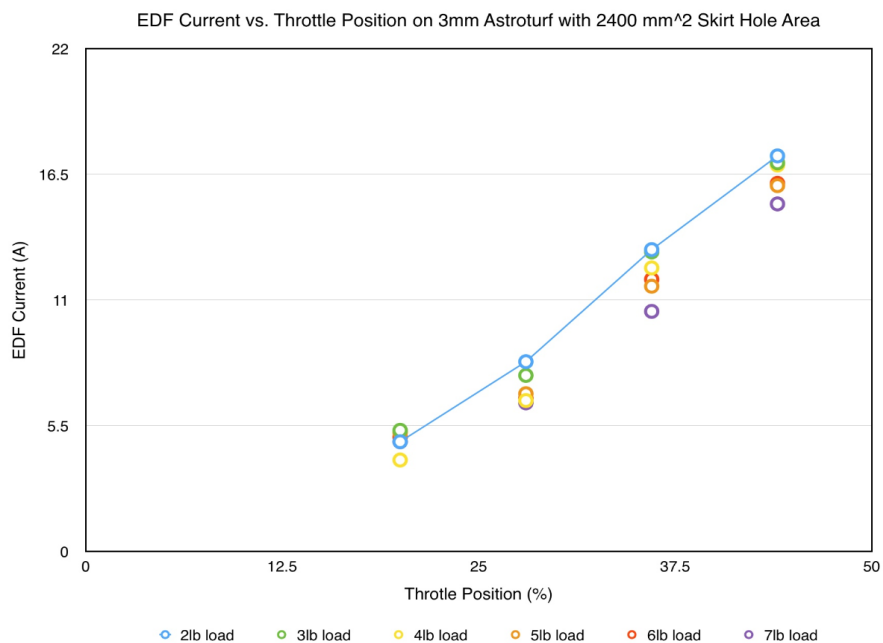


Figure 38: EDF Current vs. Throttle 1/8 in, 2400mm²

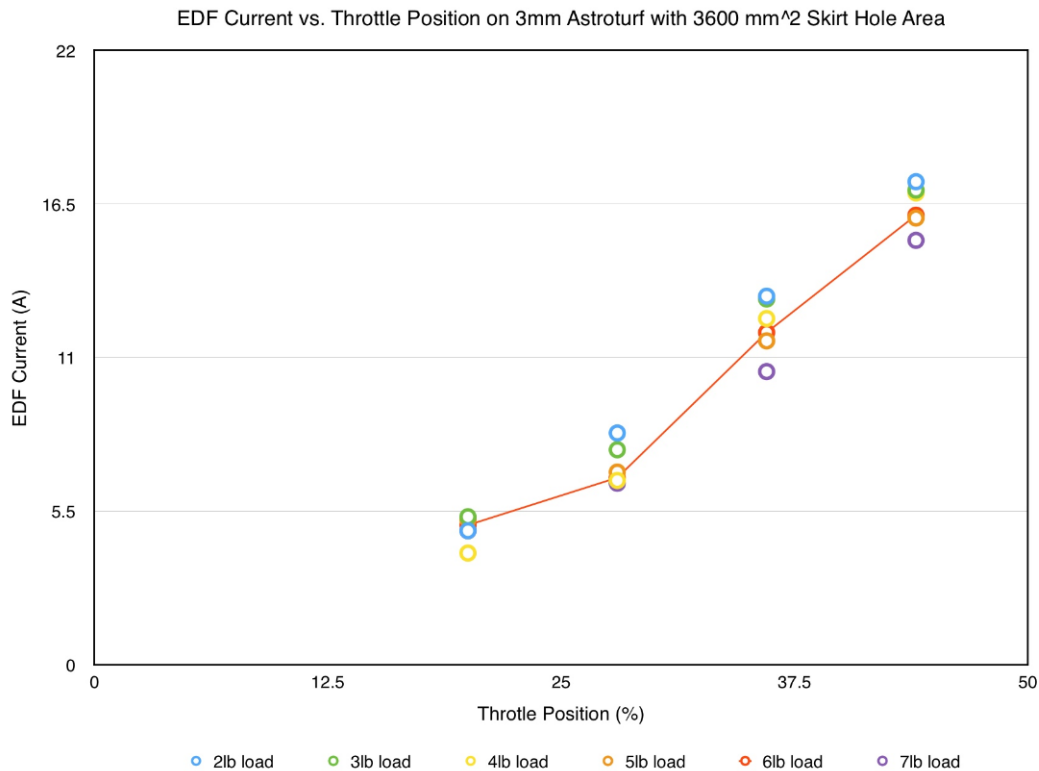


Figure 39: EDF Current vs. Throttle 1/8 in, 3600mm²

8.7.4 Economic Analysis

We anticipate that the design will fit comfortably within the allotted budget for the project. Currently, our projected budget puts the total design at approximately \$667.00. The largest category of spending is the rear thrust assembly. This assembly cost \$292.07, and comprised 43.8% of spending. The next highest category of spending is the raw materials, costing \$126.63 and comprising 19.0% of spending. Combined, these two categories comprise about 62.8% of the spent budget. This budget accounts for extra replacement parts in the critical rear thrust assembly, and leaves \$333.00 for unanticipated purchases of replacement parts or additional components. See Figure 40 on next page for the spending percentage by category.

With the exception of some common materials, we primarily purchased components from online vendors. McMaster-Carr was used to purchase the specifically sized hardware components that were needed for our assemblies. McMaster-Carr has unmatched selection when it comes to hardware, as well as reasonable prices and fast shipping. We also made extensive use of an online vendor called “Align T-Rex Store” to purchase the R/C helicopter components that were repurposed for the rear thrust fans. We decided to purchase these components online only after ensuring that our local hobby store did not carry the parts that we needed. Although the components ordered from this vendor did not arrive very quickly, they were one of the few online hobby vendors with reasonable prices that were in the United States.

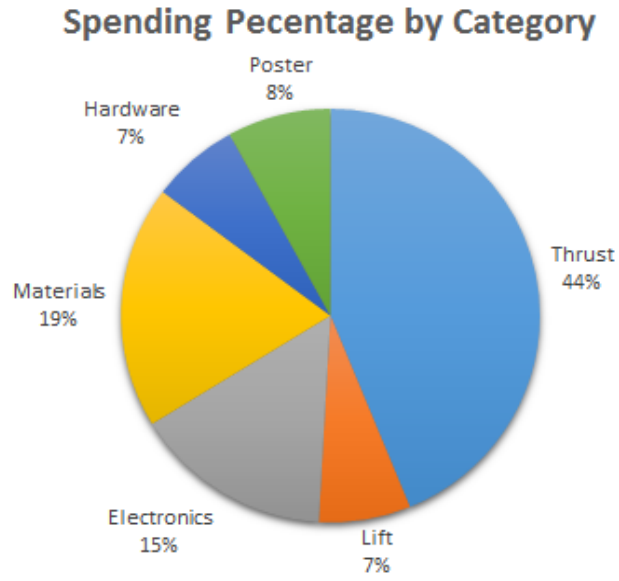


Figure 40: Spending Percentage by Category

The complete bill of materials as well as a full budget are listed in Appendix G.

9.0 Final Discussion

9.1 Construction Process

Section 8.1 details our manufacturing plan which includes a description of how to fabricate all the parts and assemble them. Note that some minor changes to the design of the hovercraft were made after this section was made. See section 9.0.1.8 for a list of all the changes to this manufacturing plan.

The images below show some of the stages of the construction process while we assembled the final hovercraft. Figure 41 pictures the skirt assembled with the lower 3 bases pieces. Figure 42 shows the gearbox being assembled. Figure 43 shows the hovercraft with the base attached. Figure 44 shows the fully constructed hovercraft.

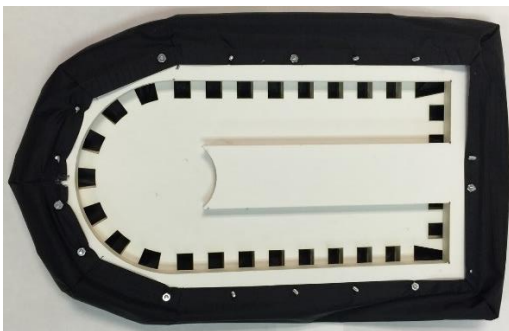


Figure 41: The base assembly before acrylic



Figure 42: Assembling the gearbox



Figure 43: Hovercraft assembled (thrust and lift)

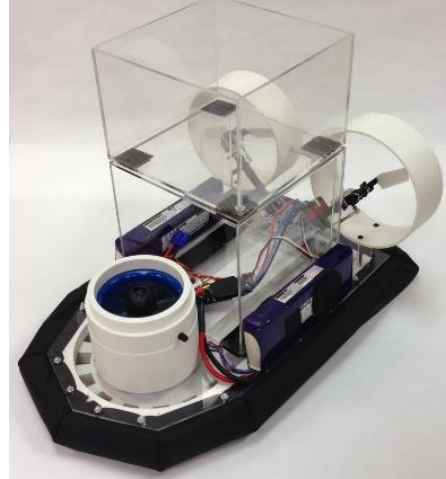


Figure 44: Fully assembled hovercraft

9.2 Testing

Final testing of the remotely piloted hovercraft was unlike a typical engineering test procedure. Instead of traveling a certain speed of performing a task with a desired accuracy and repeatability, Boeing-1 required the hovercraft to function reliably. To test this, the team performed operational tests in which a wide range of surfaces were operated on and diverse maneuvers were performed.

Throughout the testing process, many part failures were encountered and required repairs in order to continue operation. Each part failure was documented as well as the number of seconds that the hovercraft had been operating since the system was build. The lift system had been operational prior to the thrust system as the same components were used for construction of the alpha prototype. For this reason, the ducted fan for the lift system collected roughly 45 more minutes of operation time prior to assembly of the final prototype. Figures 45 and 46 show graphs of Reliability Growth of the lift and thrust systems shows the increasing trend of mean time between failures.

Boeing-1 was very pleased to see an increasing trend in the reliability growth of each system. The team would have liked to spend more time testing the final prototype in an effort to increase the mean time between failures for the thrust system to 15 minutes. The controllability and maneuverability, highlighted in the customer needs as highest priority, were met and exceeded. The hovercraft had the ability to rotate in place and reverses out of a corner, allowing for easy maneuvering through the obstacle course unassisted.

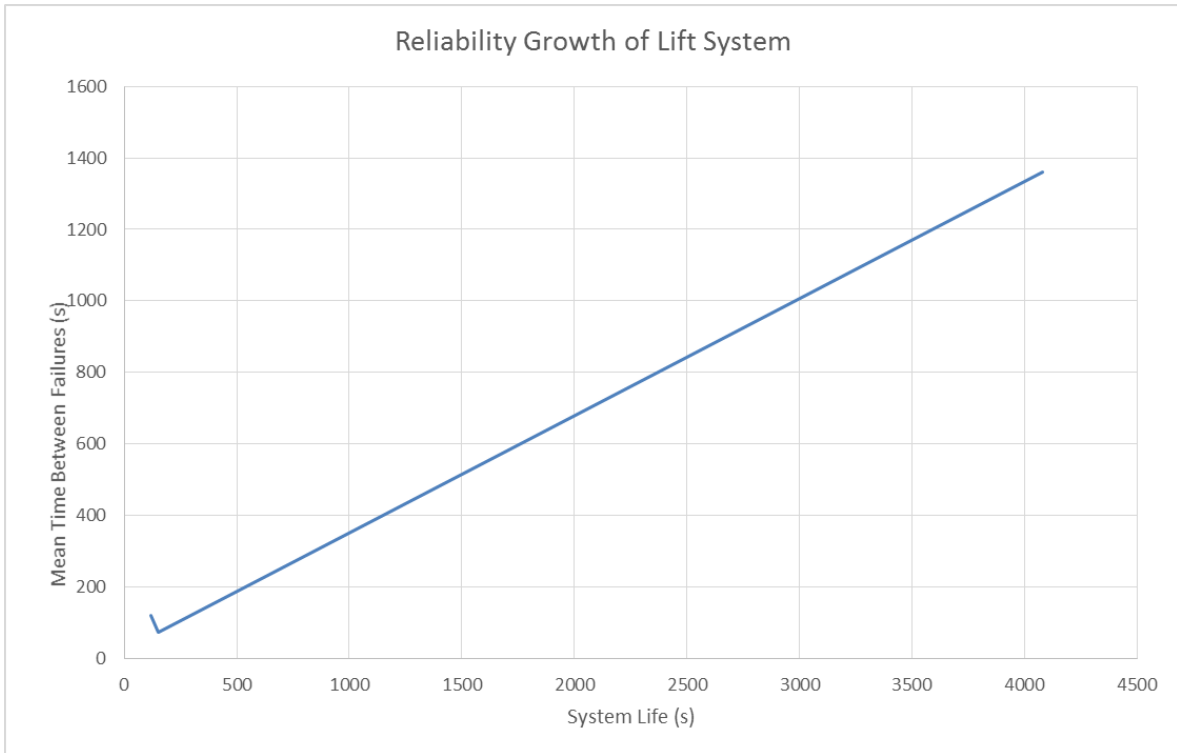


Figure 45: Reliability growth of the lift system

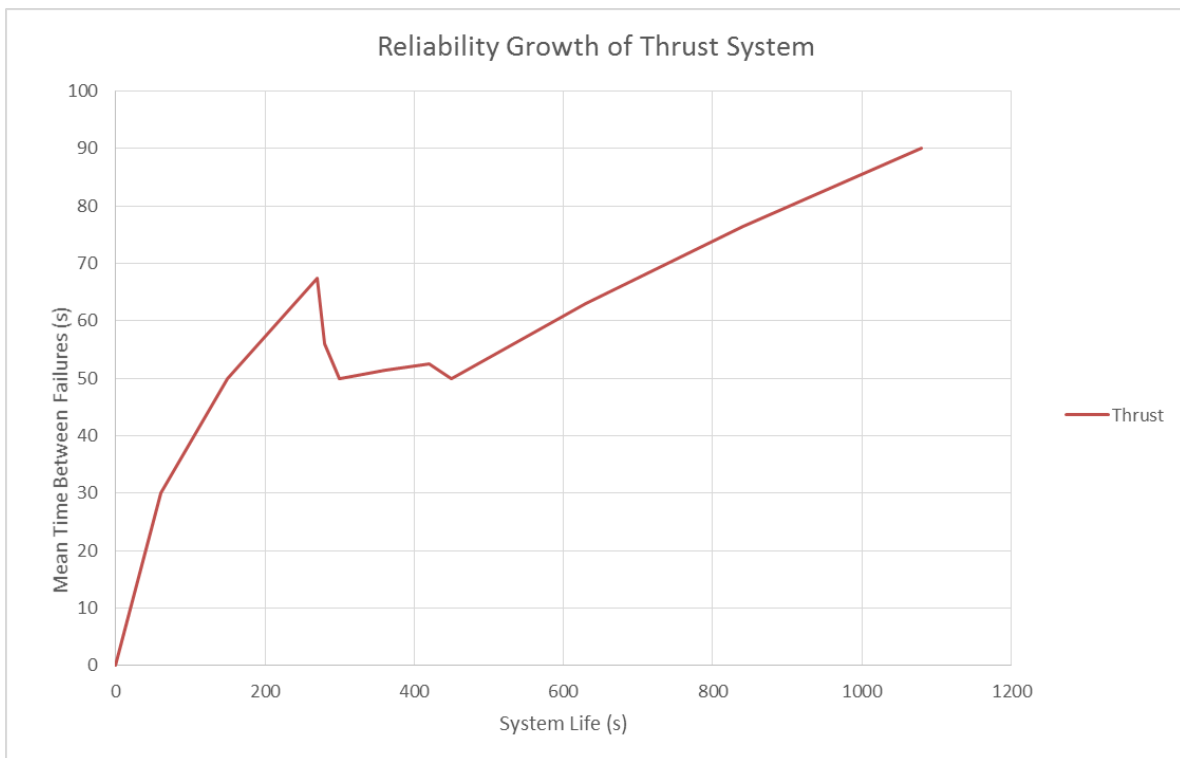


Figure 46: Reliability growth of the thrust system

Table 22: Reliability growth of the thrust assembly

Thrust				
Occurrence	Life (s)	Time Between Failure (s)	Failure	Corrective Action
1	0	0	During assembly, gearbox layers cracked where the press-fit dowel pin contacted the acrylic	Gearbox layers redesigned with thicker layers around dowel pins and bolt holes. New gearbox assembled
2	60	30	The super glue securing the jackshaft bearing to the acrylic gearbox failed. Cracks propagated around gearbox bolt heads due to contact with motor	Replaced the jackshaft in gearbox and secured with red thread locker. Gearbox layers redesigned with increased clearance between bolt heads and motor
3	150	50	Set screw securing propeller hub to shaft vibrated loose. After inspection, other thrust assembly hardware loosened	Applied thread locker to set screw and retightened. Applied thread locker to hardware and retightened
4	270	68	Left collective linkage came off of propeller hub	Replaced link
5	280	56	After inspection, left collective linkage was found to be cracked allowing bolt to pass through the link	Fabricated new, more robust, link and replaced
6	300	50	Right collective linkage came off of propeller hub	Fabricated new, more robust, link and replaced
7	360	51	Bolts securing control rod to servo arm came loose	Re-tightened
8	420	53	Bolts securing control rod to servo arm came loose	Re-tightened
9	450	50	Bolts securing control rod to servo arm came loose	Set screw did not fully engage the rod due to the screw head bottoming out on the coupling. Replaced screws with longer version
10	630	63	Thread-locker securing jackshaft bearings to acrylic gearbox failed. Jackshaft assembly shifted axially and disengaged motor pinion gear	A new gearbox was manufactured using vibration resistant Delrin for select components. Jackshaft was secured mechanically as well as via epoxy.
11	840	76	Motor pinion vibrated off of motor shaft	The pinon gear was fastened to the motor shaft again with red Loctite.
12	1080	90	Left-hand belt broke and thrust ESC solder connection broke during disassembly	The belt was replaced with one from spare parts and the solder connection was temporarily fixed. Connections were fastened poorly and the thrust motor shorted causing it to be damaged.

Table 21: Reliability growth of the lift fan

Lift Fan				
Occurance	Life (s)	Time Between Failures (s)	Failure	Corrective Action
1	120	120	Bearing vibrated loose on motor drive shaft and slid out of motor housing	Reinserted bearing
2	150	75	Bearing vibrated loose on motor drive shaft and slid out of motor housing	Reinsered bearing and secured with red thread locker
3	1080	360	No failure	NA

10.0 Conclusions and Recommendations

The objective of this project was to create a remotely piloted hovercraft that was capable of carrying a one-pound payload 6 inches above its base through an obstacle course for 15 minutes. Our group succeeded in building a hovercraft that is capable of maneuvering effectively with a payload, however we struggled with repeatability and durability. Overall we are pleased with the hovercraft we designed and built, but we were hoping for better performance from our final design.

Because we decided that maneuverability and control were the two most important design criteria, we focused most of our design time on the rear thrust system that was a promising solution for a controllable hovercraft. The ability to move forward and reverse and rotate in place made our hovercraft much more controllable than our competition's hovercraft. The Arduino code to control the rear thrust control servos was also fairly elementary and did not set us behind schedule.

Although the rear thrust system awarded the hovercraft with more maneuverability, it had a few disadvantages. One disadvantage of this rear thrust system was that it did not apply as much force to the hovercraft as we had wanted. We overestimated the amount of thrust that we could get out of the third party R/C helicopter tail rotors. Although the rotors were suitable to control a smaller model, they did not scale up appropriately for our hovercraft. Additionally, the manufacturing process for the rear thrust system was complex, and repairs were time consuming. We found that the linkages that connect our variable pitch propellers to the servos were not robust enough for rigorous testing. Since small parts like these failed frequently in our testing, much of our testing time was spent performing repairs on delicate mechanisms.

Our group identified a few recommendations that we think would improve our outcome if we had to redo the project. Firstly, we think that using two ducted fans for our thrust system would have simplified our entire design while giving us reliable strong thrust. Simplifying the design would leave more time for testing and refinements, especially in code. This time could be utilized to build a better control system for the hovercraft in software, utilizing a gyroscope or other sensors. Finally, we recommend that by testing earlier we would realize problems with our design earlier and have more time to change course.

11.0 Self-Assessment (Design Criteria Satisfaction)

After the completion of the project both the team members and the project sponsor were pleased with the resulting prototype, the gearbox, variable pitch propellers and multilayer base for air distribution being the main components highlighted after the design. Table 23 shows how most of the specified need were met by the team. Additionally as seen in Figure 1: Weighted AHP values, the items that performed the best were the ones meant for maneuverability and control, which in turn were the ones given the major importance. This shows that the team stayed true to the design objectives. On the other hand, time constraints and complexity of design limited the amount of testing that could be done on the reliability of third party components as a result the durability of the design was highly compromised as can be seen in tables 23 and 24.

Table 23: Target Specifications Self-Assessment

Met Specification?	Need #	Metric	Imp.	Units	Marginal Value	Ideal Value
YES	1,4	Payload container can hold at least 1 pound	5	lb.	1	5
YES	2,4	Payload container is 6 in above the deck	5	in	8	6
YES	3,4,12	Payload container allows for easy removal of cargo	4	s	2	1
YES	3,4	Payload container is water-tight	2	in ³ /s	1	0
YES	3,4,5	Payload container is made of smooth material	4	microns	0.12	0.03
YES	3,4,7	Payload container is at least 6in x 6in x 4in	5	in	6 x 6 x 4	6.25 x 6.25 x 4.25
YES	6	RPH dimensions doesn't exceed 18in x 12in x 12in	5	in	18 x 12 x 12	12 x 10 x 10
YES	6,12	RPH base may not exceed 8in in width	5	in	8	8
YES	6,15	Maximum inflated Skirt Thickness	2	in	5	1.5
YES	6,12	Maximum base length	3	in	18	12
YES	6,1	Maximum base Thickness	3	in	1	0.25
YES	8,9,21	RPH instills pride through design features	1	yes/no	no	yes
YES	10	Power is stored in at least one electrical battery	2	unit	3	1
YES	11	RPH development cost is at most 1000 USD	4	USD	1000	500
YES	12	New user can learn to drive the RPH within 15 minutes	3	min	15	1
NO	13	RPH can operate on water	4	mi/h	5	15
YES	12,14	RPH remains operational after going over obstacles	4	yes/no	yes	yes
YES	12,16	RPH performs on carpet and astro-turf surfaces	4	mi/h	10	20
YES	12	RPH has a sharp radius turn radius	4	in	24	0
YES	12	RPH battery is changed fast	4	s	60	5
YES	12	RPH battery/payload can be changed without use of tools	5	yes/no	no	yes
YES	12,13	RPH doesn't flip over when going over 20 degree slopes	5	yes/no	*stationary	yes
NO	17,12	RPH runs for at least 15 minutes none stop	4	yes/no	*battery change	yes
YES	18	The RPH uses the Spektrum DX7 controller	3	yes/no	yes	yes
YES	19	The RPH is controlled using wireless protocol	5	yes/no	yes	yes
YES	20,21	Dangerous mechanical components have safety features	3	yes/no	no	yes
YES	20,21,9,8	The RPH looks safe	3	yes/no	no	yes
NO	9,8,21	Wires are hidden from view during operation	2	% of wire exposed	100	10
YES	21,22	Wire connections have been insulated	3	yes/no	no	yes
YES	12	The RPH can come to a stop quickly	4	s	10	1

11.1 Customer Needs Assessment

As shown below on table 24 it can be seen that the final prototype design meet most of the requirements with great satisfaction, especially the PID control worked very well after being implemented independently of the RC controls. This is with the exception of the following 3 Instances:

- 1. The hovercraft remains stable in water and slopes:** The base design for the hovercraft was not tested in slopes or water in the early stages of prototyping, this cause the team to overlook that the current base design was not suitable for going up slopes because once the hovercraft started going into the ramp it would leave a large gap between the floor and itself, this gap caused all the air through flow through it and as a result the hovercraft lost all its hover cushion.
- 2. The hovercraft will be adaptable for changing load:** This feature was not pursued because after further validation of our system requirements the team arrived at the conclusion that the adaptable payload bin would not be the most effective approach. The reason for this is that the initial specifications given by the customer only restricted the payload to a specific volume, so changing the volume of the payload to be adjustable would not only add an extra level of complexity to our already complex design but it would also provide no advantage during the competition.
- 3. The hovercraft will run continuously for 15 minutes:** This problem was caused mostly by the manufacturing parts ordered from third party suppliers. Even though they were being used for their intended purpose, the supplier parts kept failing over and over again, and some modification to the original parts had to be made. Additionally the delay in the approval of the bill of materials cause the construction process to be delayed a week which otherwise would have helped identify some of this failures during testing.

Table 24: Customer Needs Assessment

No		Need	Need Assessment (1-10)
1	Payload	The RPH will support a 1 pound load while remaining stable.	10
2	Payload	The Payload will be secured in a structure standing 6 inches above the deck.	10
3	Payload	The load securing system will be adaptable to the changing load.	1
4	Payload	The RPH will have a container to carry the load	10
5	Payload	Payload container walls are smooth	10
6	Size	The RPH will fit within a volume of 15in x 12in x 8in	10
7	Size	The payload container fits within a volume of 6in x 6in x 4in	10
8	Aesthetics	RPH has an appealing design.	10
9	Aesthetics	Instills pride	10
10	Power Distribution	The RPH will store power in at least one of the supplied batteries.	10
11	Cost	Project costs must remain under \$1000.	10
12	Maneuverability	The RPH will be easily maneuverable	10
13	Maneuverability	The RPH will remain stable and functional in water and on slopes	1
14	Maneuverability	The RPH must be able to move to avoid obstacles	10
15	Maneuverability	The RPH is Capable of going over obstacles	8
16	Maneuverability	The RPH performs well on carpet and astro-turf	10
17	Operation	The RPH will be able to operate continuously for at least 15 minutes.	2
18	Control	The RPH will be controlled using the Spektrum DX7 controller.	10
19	Control	The RPH is controlled using wireless protocol	10
20	Safety	Propulsion and lift systems will have guarding around all fans	10
21	Safety	The RPH has to be considered Safe by Sponsors prior the race	10
22	Safety	Electrical systems have to be safe	9

11.2 Global and Societal Needs Assessment

During design and manufacturing, our team made an effort to comply with all safety regulations with the tools and materials that we utilized. Since we used the laser cutter in the learning factory, we often had to choose materials that could be cut without releasing dangerous fumes that could harm us or those around us. We also kept safety in mind in our design by guarding the main lift fan and rear thrust fans from outside objects.

Besides safety in manufacturing and design, the nature of our project did not specifically address any more general global and societal needs. Nevertheless, the design principles that we utilized over the course of the project could be endlessly applied to projects with greater global impact. We believe that choosing materials that meet safety specifications is simply just useful practice for choosing safe environmental materials for projects on a larger scale. Because we valued safety in our design and learned valuable industry skills, we assess our own performance in addressing global and societal needs to be 7/10.

References

1. Andrew, Labat Ivan Pierre. "Patent US3589058 - Toy Ground Effect Vehicle with Adjustable Stabilizing Weight." Google Books. USPTO, 29 June 1971. Web. 23 Sept. 2014.
2. Blum, Klaus. "Patent US6591928 - Hovercraft." Google Books. USPTO, 15 July 2003. Web. 23 Sept. 2014.
3. Hetman, Michael. "Patent USD564046 - Set of Air Cushions for Hovercraft Toy Vehicles." Google Books. USPTO, 11 Mar. 2008. Web. 23 Sept. 2014.
4. Lee, Chun Wah, Robert Spalinski, Matthew Del Duke, and Justin Discoe. "Patent US7032698 - Hovercraft." *Google Books*. USPTO, 25 Apr. 2006. Web. 23 Sept. 2014.
5. Mercier, Michael. "Patent US20130333968 - Directional Control System for Hovercraft." *Google Books*. USPTO, 19 Dec. 2013. Web. 23 Sept. 2014.
6. Tilbor, Neil, and Michael Hetman. "Patent USD451560 - Hovercraft Toy Vehicle." *Google Books*. USPTO, 4 Dec. 2001. Web. 23 Sept. 2014.
7. Tinajero, Anibal. "Patent US3279553 - Pitch and Roll Control for Ground Effect Machines." Google Books. USPTO, 18 Oct. 1966. Web. 23 Sept. 2014.
8. "T-Rex 25 Instruction Manual." *T-Rex-parts.com*. Align Corporation Limited, 1 Nov. 2008. Web. 28 Oct. 2014"

Appendix A: Project Management

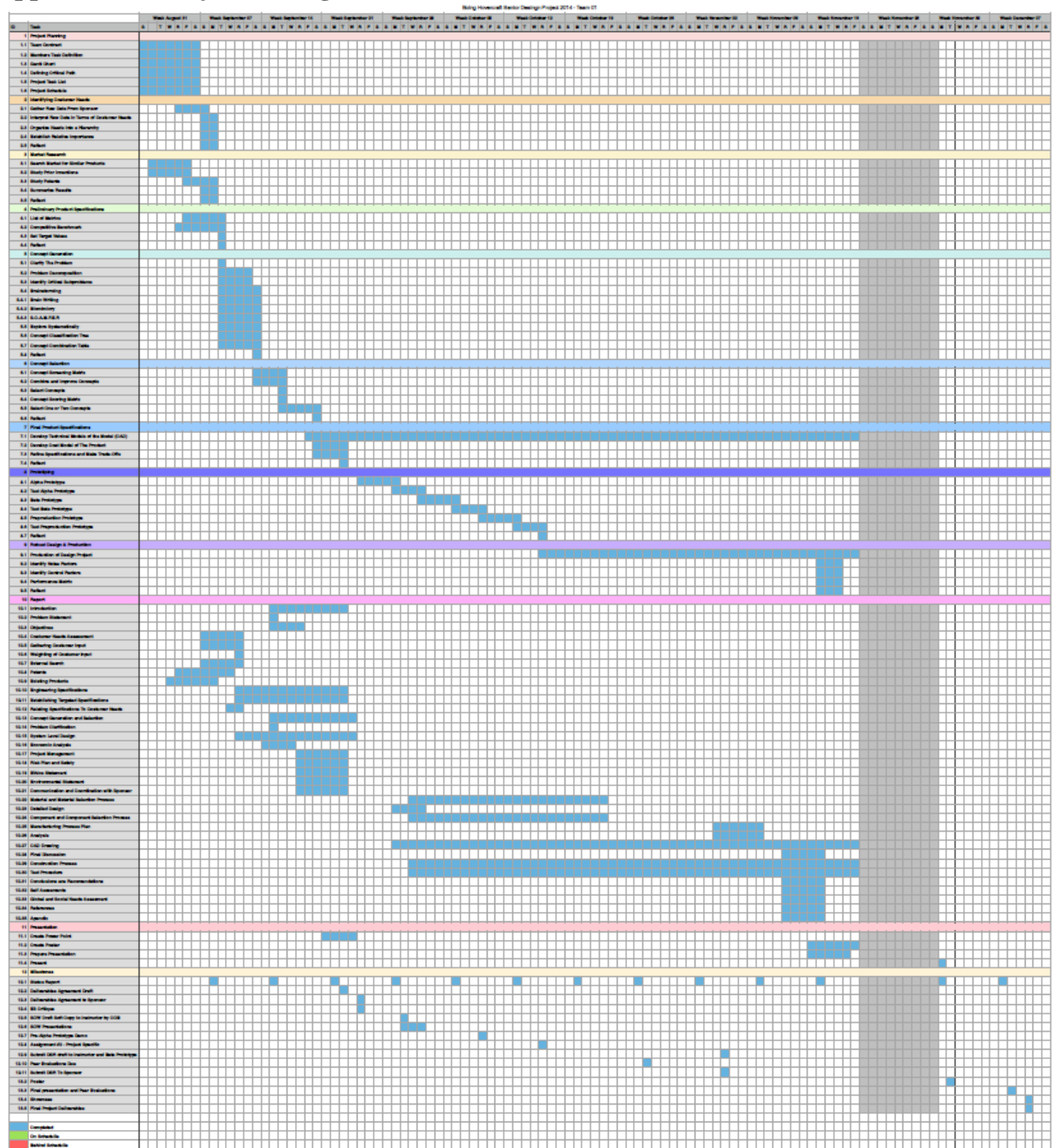


Figure 47: Gantt Chart (as of 9/20/14)



Appendix B: Full Patent Research

1: Pitch and roll control for ground effect machines

Patent Number: US 3279553 A

Filing Date: March 23, 1962

Inventor: Anibal A. Tinajero

Patent summary:

US patent 3279553 A, filed on March 23, 1962 claims the right to specific design methods for continually controlling the pitch, roll, and yaw of a ground effects vehicle. A ground effects vehicle is defined as a vehicle that travels through level flight near the earth's surface by means of aerodynamic interaction with the surface. The specific means for controlling the motion of the hovercraft include four nozzle sections, one located on the forward, aft, port, and starboard sides of the vehicle. Each nozzle section contains movable vanes that direct the direction of the airflow that provides floatation to the vehicle as well as the rate of airflow. The directional vanes control the yaw, rotation of the vehicle about a vertical axis through the center of gravity of the vehicle. The variable flow rate allows for control of the vehicle's pitch, tilt fore and aft, as well as roll, tilt left and right.

The patent also claims rights to the specific method of controlling the vanes and flaps for flow direction and mass flow rate, respectively. The vanes, being hingedly attached to the vehicle body are independently varied for each nozzle section by a servo-controlled rod attached to the vane at a distance from the hinge point. The flow rate is controlled in a similar manner, however, flaps are adjusted to control the mass flow rate of air flowing through the directional vanes to cause different amounts of lift force on different sides of the vehicle, therefore achieving a pitch or rolling motion.

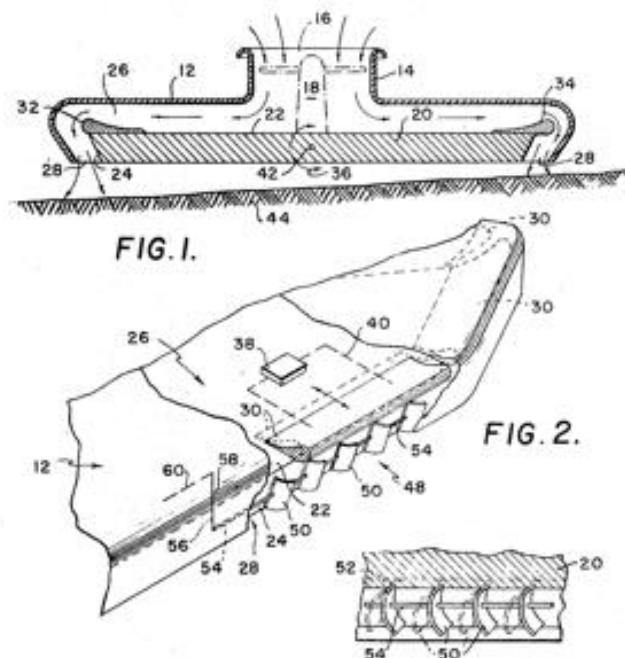


Figure 49: Detailed Drawings 1

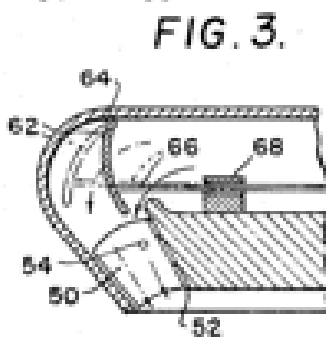
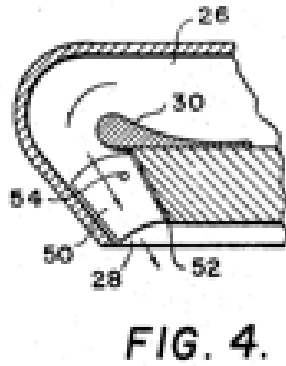


Figure 50: Detailed drawings 2

Ramifications for Boeing RC Hovercraft design:

This patent outlines some very useful and simple methods for controlling pitch, roll, and yaw of a hovercraft. Since this patent was filed in 1962 it will not present an obstacle to design for the project as the patent has since expired. The purpose for listing it is to outline a possible method for vehicle control and not to highlight a design that is to be avoided for legal reasons. The method may be best implemented if used partially for its proposed pitch/roll control and not for its yaw control as other methods can be used for turning about a vertical axis.

2: Hovercraft

Patent Number: US 7032698 B2

Filing Date: October 19, 2004

Inventors: Chun Wah Lee

Robert Paul Spalinski

Matthew James Del Duke,

Justin Discoe

Chow Ming Lau

Patent summary:

The patent covers hovercraft of any size using the claimed design. The background section clearly states that the hovercraft may be full scale or miniaturized. There are three main components, which the patent claims. One is a thrust-lift fan, which is mounted primarily in the for-aft direction. However, the fan is directed downward into the hull of the craft so that it can provide both lift and thrust. At the rear exit of the fan dust about half of the outlet is directed into the hull of the craft and the other half is usable for thrust. Image 1 shows a clear view of the lift-thrust fan. The second claim of the patent is a fixed steering fan that is mounted toward the front of the craft with axis transverse to the lengthwise axis of the craft. This motor can be operated forward or backward depending on which direction of craft rotation is desired. Figure 42 also shows a clear view of the steering fan. The third and final primary claim made in the patent regards a guide wheel, which is mounted at the rear of the craft and, via a spring system, floats along the ground level to provide support for rotation in the horizontal axis. The guide wheel suspension mounts to the hull of the craft and the wheel provides a pivot point that the craft can rotate around when turning. Figure 43 shows the underside of the craft and the location of the guide wheel at the rear of the craft and Figure 44 shows a diagram of the wheel and its suspension system.

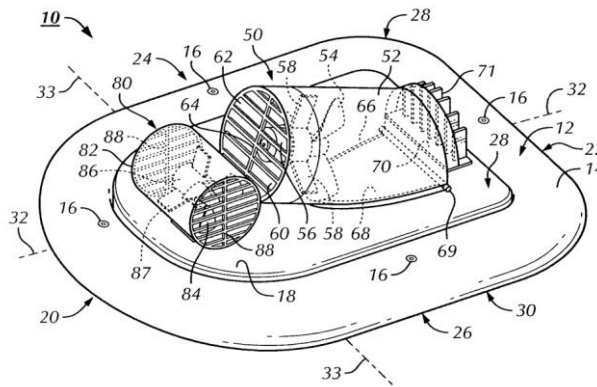


Figure 51: Thrust-lift fan and steering fan

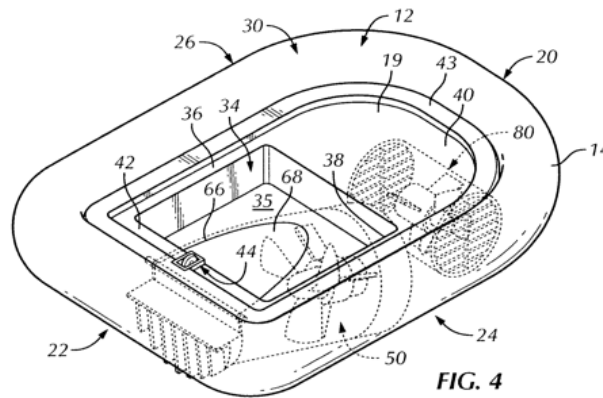


Figure 52: Guide wheel labeled as #44

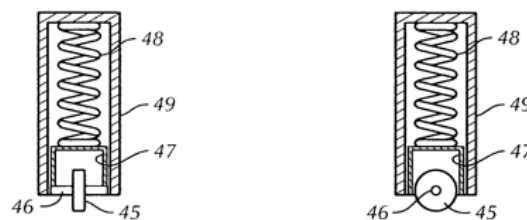


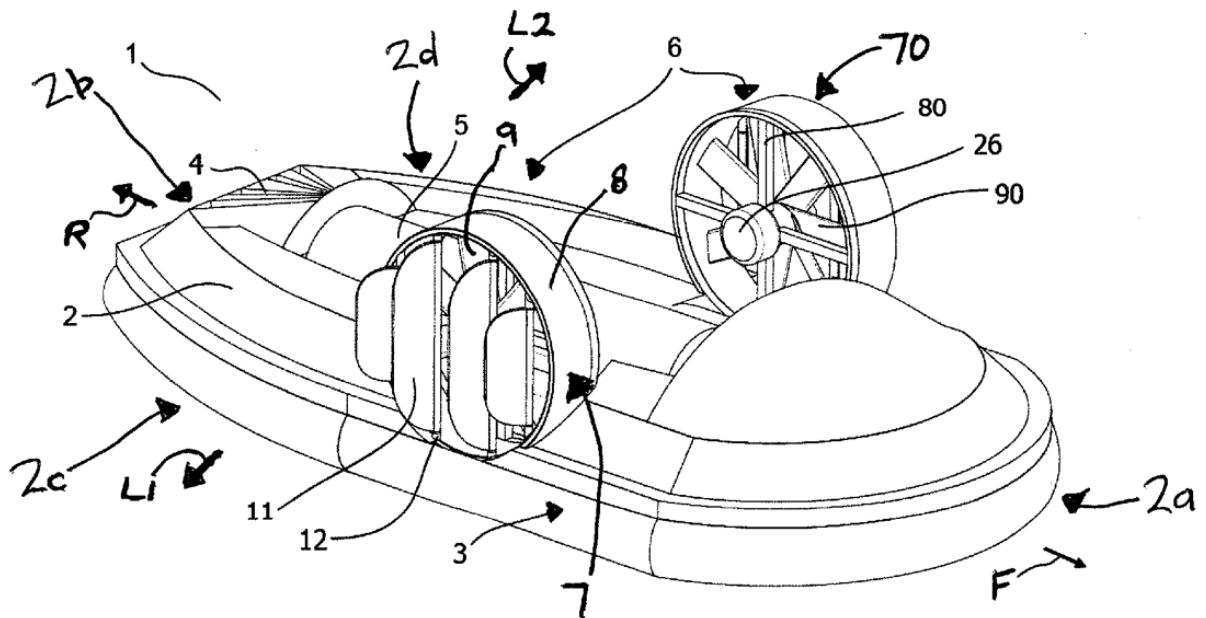
Figure 53: Guide wheel suspension

Ramifications for Boeing RC Hovercraft design:

The following patent presents design limitations for the design for the Boeing Hovercraft Senior Design project. Since the patent was filed in 2004, less than 20 years ago, its claimed material is still protected. This means that while developing concepts for the RC hovercraft, the design team must make sure to not infringe on this patent. Claimed material includes a thrust-lift fan, a fixedly mounted transverse steering fan, and a spring suspended guide wheel to aid in turning. The guide wheel will not be a problem to avoid as the Boeing specifications state that no wheels are allowed to contact the ground. The most difficult will be avoiding the use of a transversely mounted

Inventor: Michael William Mercier

The following patent claims rights to a directional control system for an air cushion vehicle comprised of two fans with their axes mounted transverse to the fore/aft direction of the craft. Airflow from these fans can be directionally controlled via rudder and directed either for, aft, or to the side that they are facing. Figure 45 shows how the two fans are mounted on the sides of the craft facing outward so as to provide quick directional control. Figure 46 shows how directional vanes can be used to direct the flow from both fans toward the rear of the craft to generate a net forward thrust while canceling out the side thrust causing the craft to move straight forward.



68

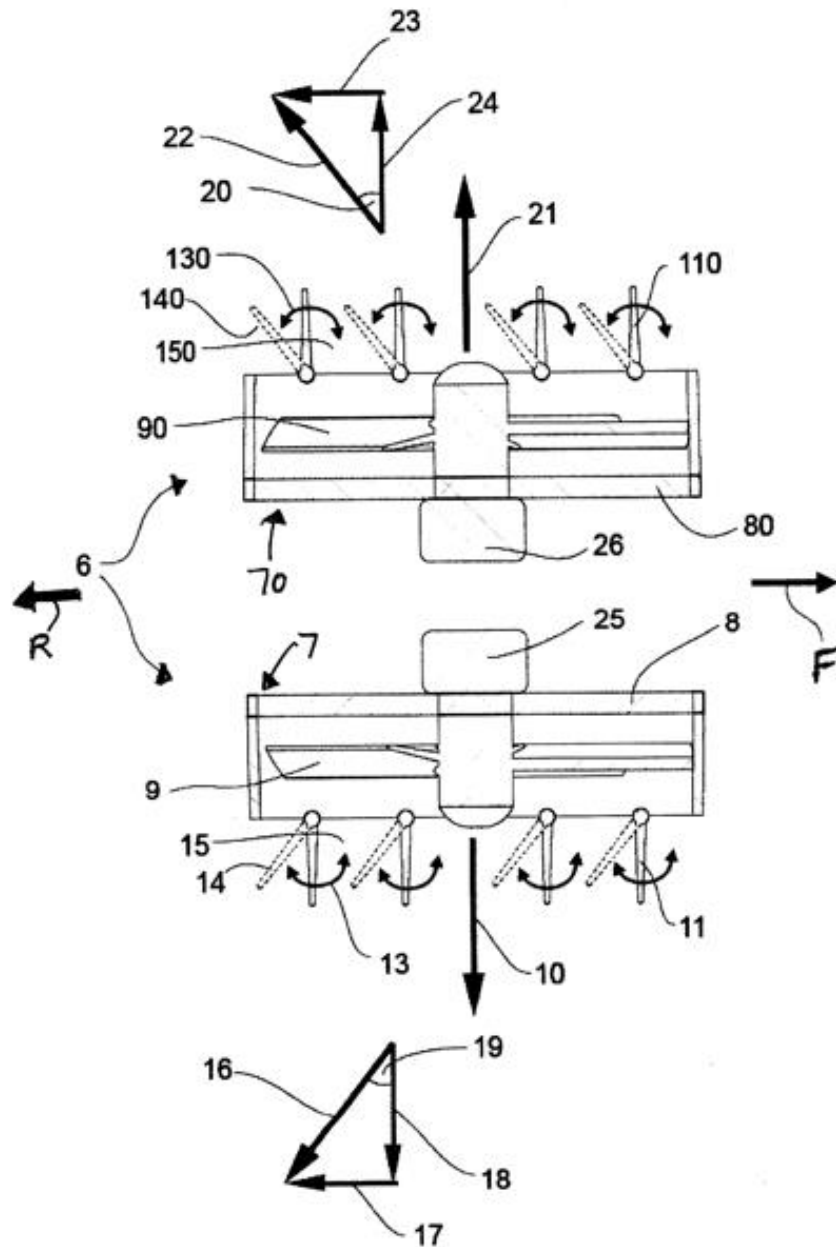


Figure 55: Generation of net forward thrust

Ramifications for Boeing RC Hovercraft design:

The above patent poses limitations on the design of the Boeing RC hovercraft as it was filed in 2013 and remains valid. When designing a control system for the Boeing RC hovercraft the team must be sure to avoid any scheme that will involve outward facing side fans that can be directed partially in the rearward direction to create a net forward thrust. This limitation may pose challenging as it severely limits the freedom to orient side thrusters in the event that more powerful direction control is required.

4: Hovercraft Toy Vehicle

Patent Number: US D451560 S1

Filing Date: April 24, 2001

Inventors: Neil Tilbor

Michael G. Hetman

Patent Summary:

The patent below claims the rights to the ornamental design of the toy hovercraft shown in the Figure below. The design consists of a craft with platform of curving leading edge, straight sides and a straight trailing edge. The craft appears to be propelled by two rear-mounted thrust fans housed in a protective cage.

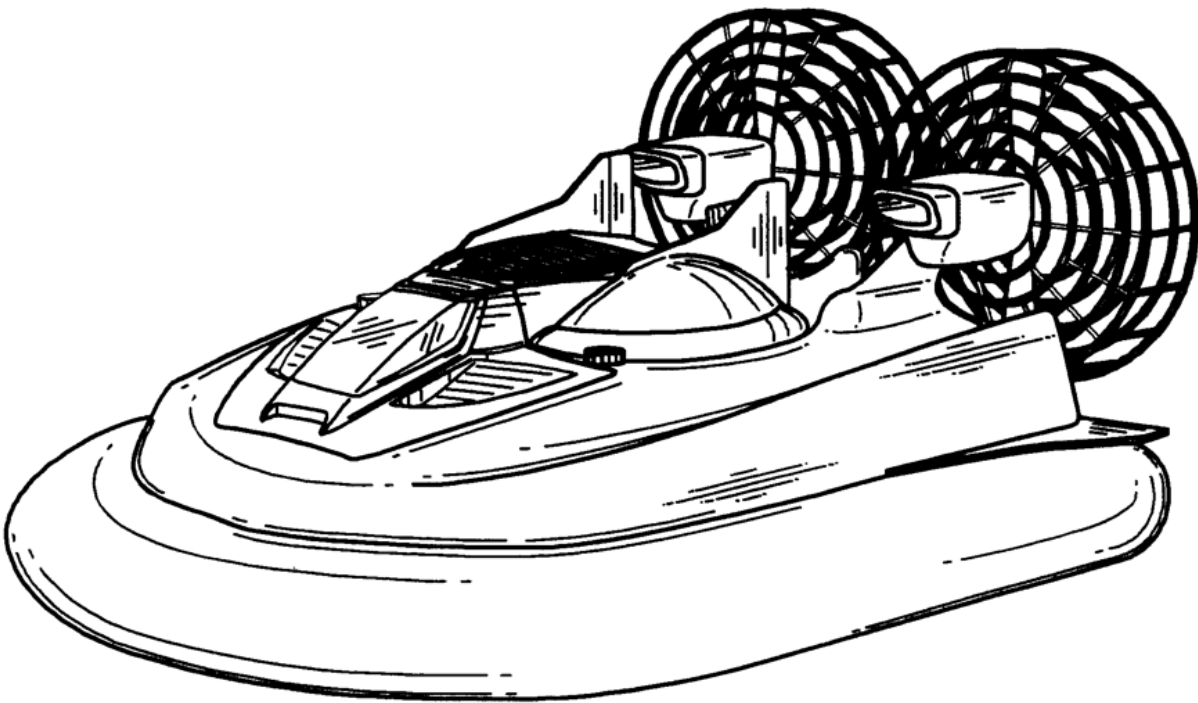


Figure 56: Hovercraft Ornamental Design

Ramifications for Boeing RC Hovercraft design:

The previous patent poses few design challenges to the Boeing RC Hovercraft Design Team. As only the ornamental design is claimed in the patent, it can easily be designed around. This is especially easy since the final product of the design challenge must not be a marketable product and must only perform well in an obstacle course. For this reason, aesthetics will be of little importance during the design process.

5: Air Cushions for Hovercraft Toy

Patent Number: US D564046 S1

Filing Date: March 28, 2006

Inventors: Michael G. Hetman

Patent Summary:

The patent below claims the rights to the ornamental design of a set of air cushions for a toy hovercraft shown in Figure 48. The design consists of a craft with two supporting air cushions that provide floatation on uneven ground surfaces.

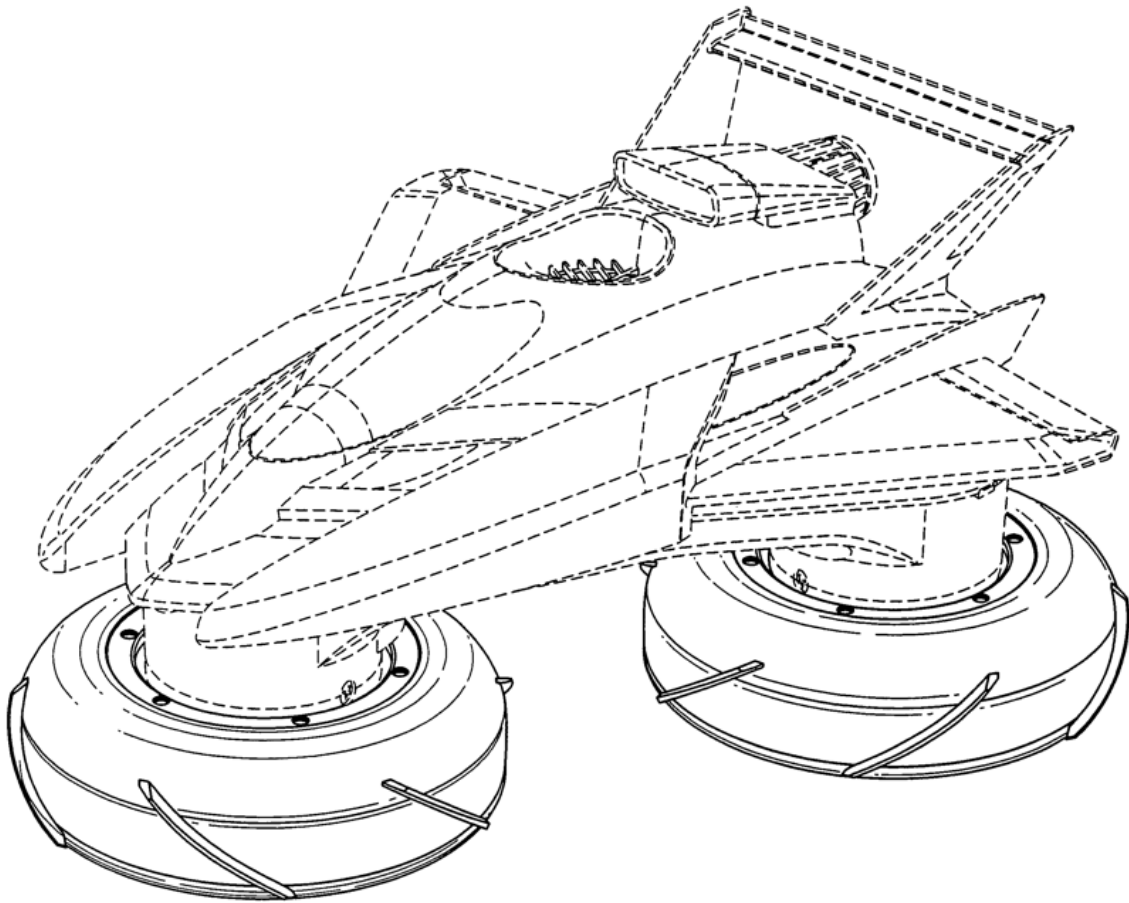


Figure 57: Air Cushion Ornamental Design

Ramifications for Boeing RC Hovercraft design:

The previous patent poses few design challenges to the Boeing RC Hovercraft Design Team. As only the ornamental design is claimed in the patent, it can easily be designed around. The team must be sure to avoid using a set of two circular air cushions in its final design.

6: Adjustable stabilizing weight

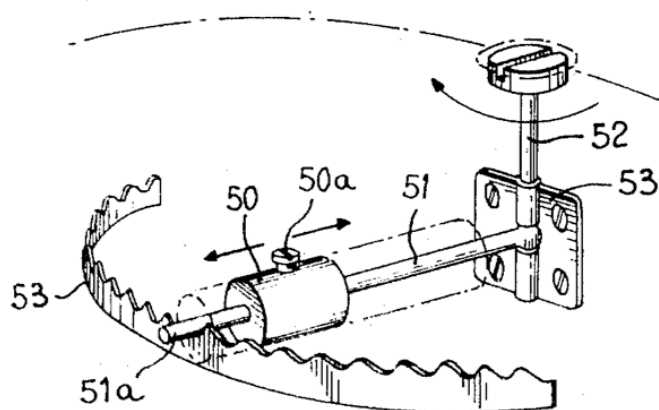
Patent Number: US 3589058 A

Filing Date: December 18, 1969

Inventors: Labat Ivan Pierre Andrew

Patent Summary:

The following patent outlines a method for adjusting a stabilizing weight for a hovercraft vehicle. The hovercraft is defined as having a hull with a skirt and at least one lift fan and one thrust fan. The fans' propellers are said to be made of flexible material so that they extend outward when the electric motors are operated. The shafts are connected to the motor by magnetic torque limiting couplers. The primary claim of the patent, however, is for the method by which the movable ballast weight is adjusted. The patent outlines that the weight will be able to slide on a shaft, and that shaft will rotate axially about a fixed end effectively moving the weight onboard the hovercraft. Figure 49 better describes how the weight will be adjusted by including an image.



INVENTOR:
IVAN P.A. LABAT
By
Breitefeld & Levine
ATTORNEYS

Figure 58: Adjustable ballast weight

Ramifications for Boeing RC Hovercraft design:

The patent above claims the rights to a method of adjusting movable ballast weight on a hovercraft. The method is usable in design, however, since the patent was filed in 1969. The Boeing RC Hovercraft design group may want to use the idea and apply active electronic control to it so that the movable ballast could be used to actively control the pitch and roll of the craft, possible even using the batteries as the ballast weight to cut down on extra material.

7: Hovercraft Directional Controls

Patent Number: US 6591928 B1

Filing Date: October 16, 1998

Inventor: Klaus Blum

Patent Summary:

The following patent claims rights to three different schemes for controlling directional travel of a hovercraft. All configurations include the use of a directionally controllable thrust fan that changes directions by means of pivoting the fan body about a vertical axis. The first configuration is depicted in Figures 50 and 51, in which two rotatable thrust fan units are mounted on the sides of the hovercraft symmetrically about a lengthwise axis to the vehicle. The second configuration is shown in Figure 52, where the two main propulsion fans are mounted rigidly at the rear of the vehicle symmetric about the lengthwise axis and a third directionally changeable thrust fan is mounted between them to alter the direction of the travel of the vehicle. The third configuration is shown in Figure 53, where two thrust/steering fans are mounted along the central lengthwise axis of the vehicle. One primary propulsion fan is mounted at the rear and a secondary steering fan is mounted on the bow, providing a moment on the vehicle to alter direction. The first and third configurations allow for movement of the vehicle in a direction that the vehicle is not directed.

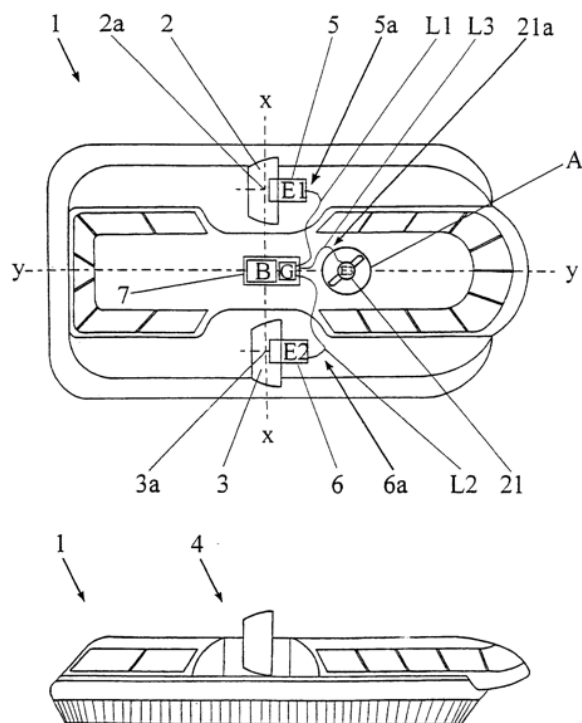


Figure 59: Configuration #1 thrusting in forward direction

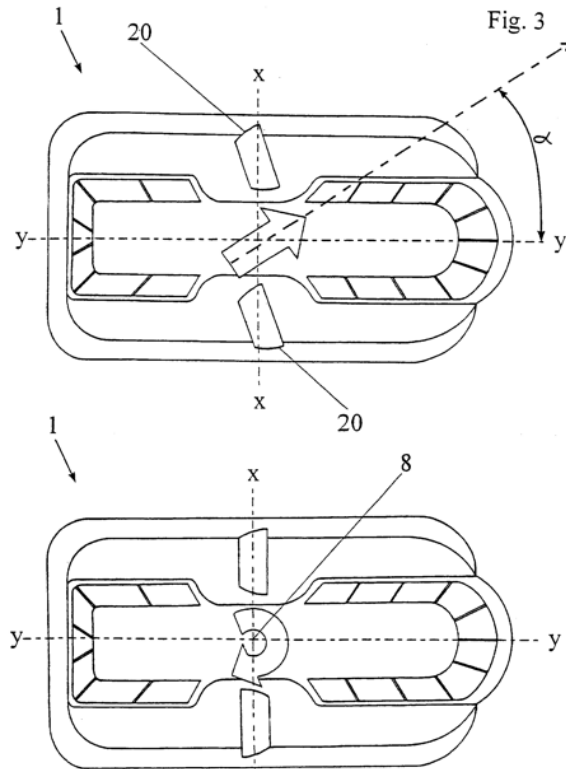


Figure 60: Configuration #1 thrusting at angle and rotating

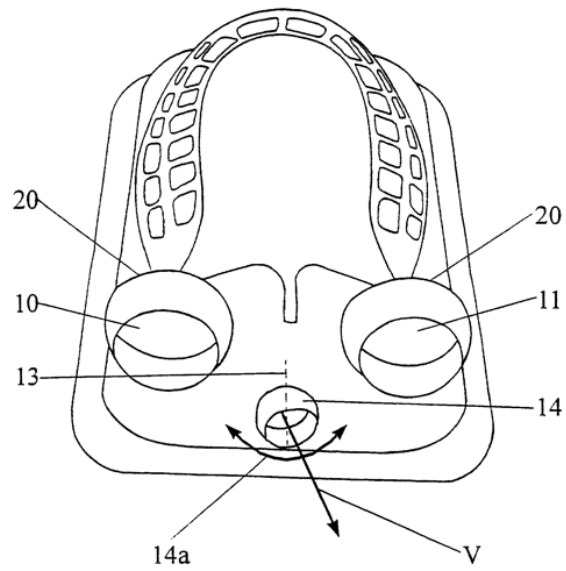


Figure 61: Configuration #2

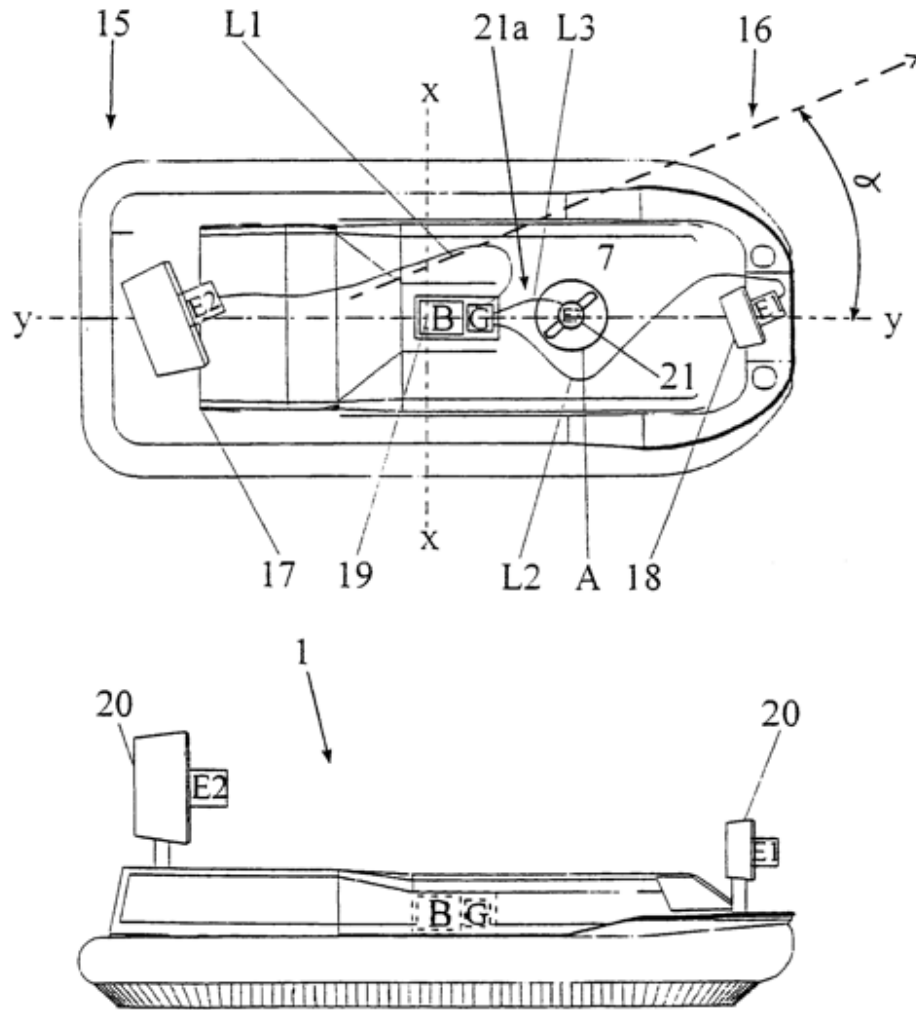


Figure 62: Configuration #3

Ramifications for Boeing RC Hovercraft design:

The three propulsion/steering configurations outlined in the above patent will present challenges for the design team, as some of the schemes are very enticing for the remotely piloted hovercraft. Boeing-1 will need to be certain not to infringe on any of the claims made in the above patent including having rotatable fan units that are used for directional control of a hovercraft.

Appendix C: Concept Combination Trees

Combination Tree Concept M

Solutions To Subproblem Of Steering The RPH

- Differential speeds on propulsion fans
- Rotating propulsion fans
- Rudders
- Directional fans on the sides
- One fan facing each direction of motion
- Rotating Mass
- Fan pattern

Solutions To Subproblem Of Skirt Design

- Fingers
- Multiple skirts
- Toe nails (friction free material on skirt)
- Bag and fingers
- Bag skirt
- Doughnut Skirt
- Protective Material added to skirt
- Pre-inflated Skirt
- Irregular Shape Skirt

Solutions To Subproblem Of Payload

- Mass supported by springs
- Reverse Pendulum
- Threaded road and motor 1D
- Threaded road and motor 2D
- Belt system and motor 1D
- Belt system and motor 2D
- Fixed payload
- Threaded road and rotating disk

Solutions To Subproblem Of Fan Position

- Front
- Middle
- Back
- Front-Back
- Front Middle
- Middle Back
- Front Middle Back

Solutions To Subproblems Of Body Design

- Square
- Circular
- Irregular
- Rounded front
- Rounded front and back
- Laminated sheets base
- Solid body
- Rounded corners

Solution to Subproblem of Propulsion-Braking

- Hinges for braking
- Fan blades reversible direction
- Nozzle
- Use air from skirt
- Compressed air tanks
- Stacked fans for differential speed
- Variable pitch propeller
- Differential speeds fans
- Fast skirt deflation mode

Solutions To Subproblem Of Inflation

- Steam
- Chemical
- Unique Fan
- Multiple Fans
- From Propulsion Fans

Solutions To Subproblem Of Steering The RPH

- Differential speeds on propulsion fans
- Rotating propulsion fans
- **Rudders**
- Directional fans on the sides
 - One fan facing each direction of motion
- Rotating Mass
 - Fan pattern

Solutions To Subproblem Of Skirt Design

- **Fingers**
- Multiple skirts
- Toe nails (friction free material on skirt)
- Bag and fingers
 - Bag skirt
- Doughnut Skirt
 - Protective Material added to skirt
 - Pre inflated Skirt
- Irregular Shape Skirt

Solutions To Subproblem Of Payload

- Mass supported by springs
- Reverse Pendulum
- **Threaded road and motor 1D**
- Threaded road and motor 2D
 - Belt system and motor 1D
- Belt system and motor 2D
 - Fixed payload
 - Threaded road and rotating disk

Solutions To Subproblem Of Fan Position

- Front
- Middle
- **Back**
 - Front Back
- Front-Middle
 - Middle Back
- Front-Middle-Back

Solutions To Subproblems Of Body Design

- Square
- Circular
- Triangular
- **Rounded front**
- Rounded front and back
- **Laminated sheets base**
- Solid body
- Rounded corners

Solution to Subproblem of Propulsion-Braking

- Hinges for braking
- Fan blades reversible direction
 - Nozzle
 - Use air from skirt
- Compressed air tanks
 - Stacked fans for differential speed
- **Variable pitch propeller**
- Differential speeds fans
- Fast skirt deflation mode

Solutions To Subproblem Of Inflation

- Steam
 - Chemical
- **Unique Fan**
- Multiple Fans
- From Propulsion Fans

Solutions To Subproblem Of Steering The RPH

- Differential speeds on propulsion fans
- Rotating propulsion fans
- Rudders
- Directional fans on the sides
- One fan facing each direction of motion
- Rotating Mass
- Fan pattern

Solutions To Subproblem Of Skirt Design

- Fingers
- Multiple skirts
- Toe nails (friction free material on skirt)
- Bag and fingers
- Bag skirt
- Doughnut Skirt
- Protective Material added to skirt
- Pre-inflated Skirt
- Irregular Shape Skirt

Solutions To Subproblem Of Payload

- Mass supported by springs
- Reverse Pendulum
- Threaded road and motor 1D
- Threaded road and motor 2D
- Belt system and motor 1D
- Belt system and motor 2D
- Fixed payload
- Threaded road and rotating disk

Solutions To Subproblem Of Fan Position

- Front
- Middle
- Back
- Front-Back
- Front-Middle
- Middle-Back
- Front-Middle-Back

Solutions To Subproblems Of Body Design

- Square
- Circular
- Triangular
- Rounded front
- Rounded front and back
- Laminated sheets base
- Solid body
- Rounded corners

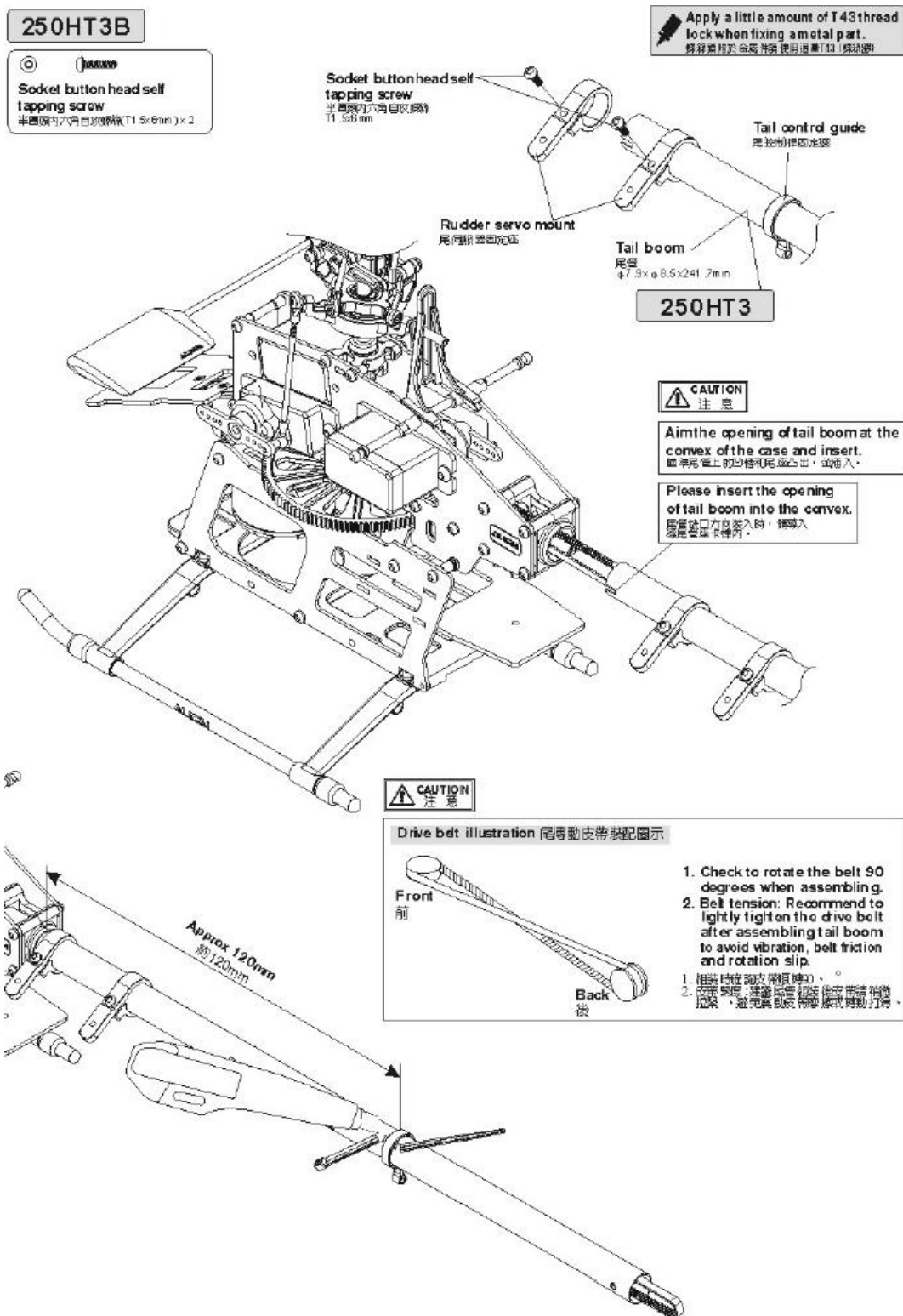
Solution to Subproblem of Propulsion-Braking

- Hinges for braking
- Fan blades reversible direction
- Nozzle
- Use air from skirt
- Compressed air tanks
- Stacked fans for differential speed
- Variable pitch propeller
- Differential speeds fans
- Fast skirt deflation mode

Solutions To Subproblem Of Inflation

- Steam
- Chemical
- Unique Fan
- Multiple Fans
- From Propulsion Fans

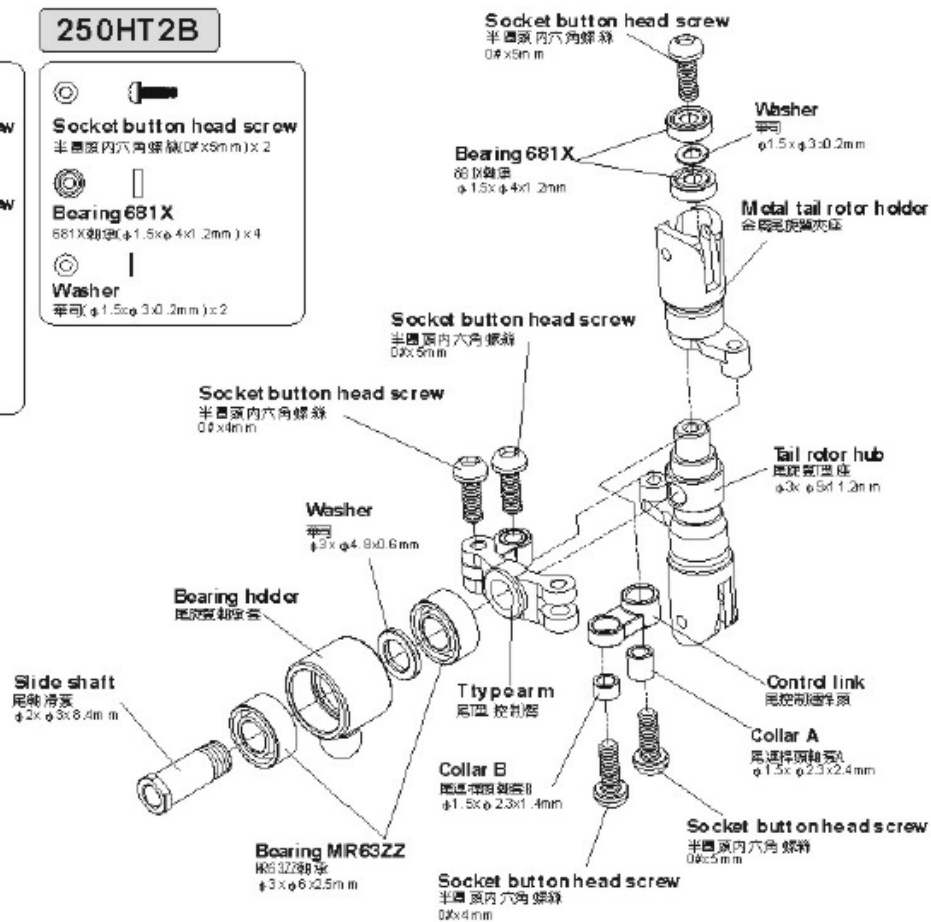
Appendix D: Align T-Rex Manual



250HT2D



250HT2B



250HT2B


 **Socket button head screw**
半圓頭內六角螺絲 (0#x6mm) x2


 **M2 Set screw**
M2止動螺絲 (M2x2mm) x1

250HT2C

 **Socket button head screw**
半圓頭內六角螺絲 (0#x8mm) x1

 **Linkage ball A(0#x2)**
取球A (0#x2) (φ3.5x5.3mm) x1

 **Apply a little amount of T43 thread lock when fixing a metal part.**
螺絲鎖用於金屬零件使用適量T43 (螺絲膠)

 **CAUTION**
注意

When tightening a screw to a plastic part, please tighten it firmly, but not over tightened, or they will strip.
螺絲鎖入塑膠件時務必注意，適量扭力鎖緊即可，而過緊的扭力可能會導致滑牙。

For original manufactory package, if the product is already assembled by Factory, please check again if screws are firmly secured and applied with some glue.
原廠零件出廠包裝如呈原裝狀態，請再確認各螺絲是否鎖緊上緊。

 **CAUTION**
注意

Arm tail rotor hub at the concave of tail rotor shaft and fix it, please apply a little glue on the set screw.
尾旋臂1/3處與尾旋軸的凹面扣鎖上，請塗點止動螺絲上緊。

Already assembled by factory, please not to check again.
已組裝完成，請務必自行再確認。

Tail pitch assembly
尾旋臂控制組

Tail blade
尾旋葉

Socket button head screw
半圓頭內六角螺絲
0#x6mm

M2 Set screw
M2止動螺絲
M2x2mm

Tail rotor control arm
尾旋臂控制臂

Collar
尾旋臂控制臂軸套
φ1.5xφ2.5x5mm

Linkage ball A(0#x2)
取球A (0#x2)
φ3.5x5.3mm

Socket button head screw
半圓頭內六角螺絲
0#x6mm

 **CAUTION**
注意

When tightening a linkage ball to a plastic part, please note to use a little CA glue and tighten it firmly, but not over tightened, or they will strip.
螺絲鎖入塑膠件時務必注意，使用少量CA膠並適量扭力鎖緊即可，而過緊的扭力可能會導致滑牙。

Care must be taken during assembly to ensure tail grips operate smoothly without binding. Any slight binding may affect tail action during flight.
組裝時，確保尾夾座滑順，些微干涉將可能導致飛行時尾動作不順暢。

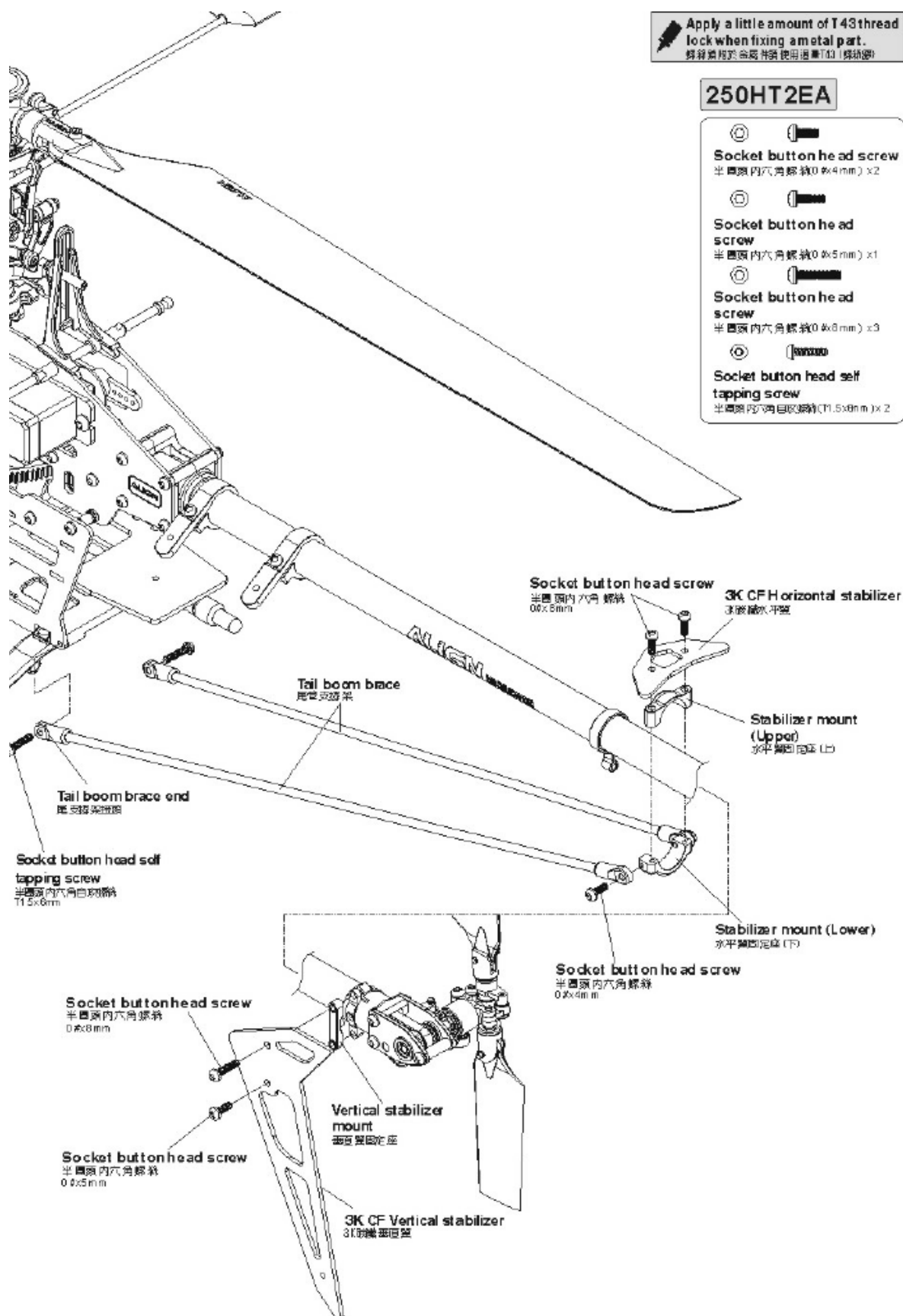
Tail pitch assembly
尾旋臂控制組

Tail pitch bell crank must be parallel to tail output shaft to ensure sufficient pitch travel range.
尾旋臂控制曲柄必須與尾輸出軸平行以確保行程量足夠。

Apply a little amount of T43 thread lock when fixing a metal part.
 螺絲鎖附於金屬件時使用適量T43(螺絲膠)

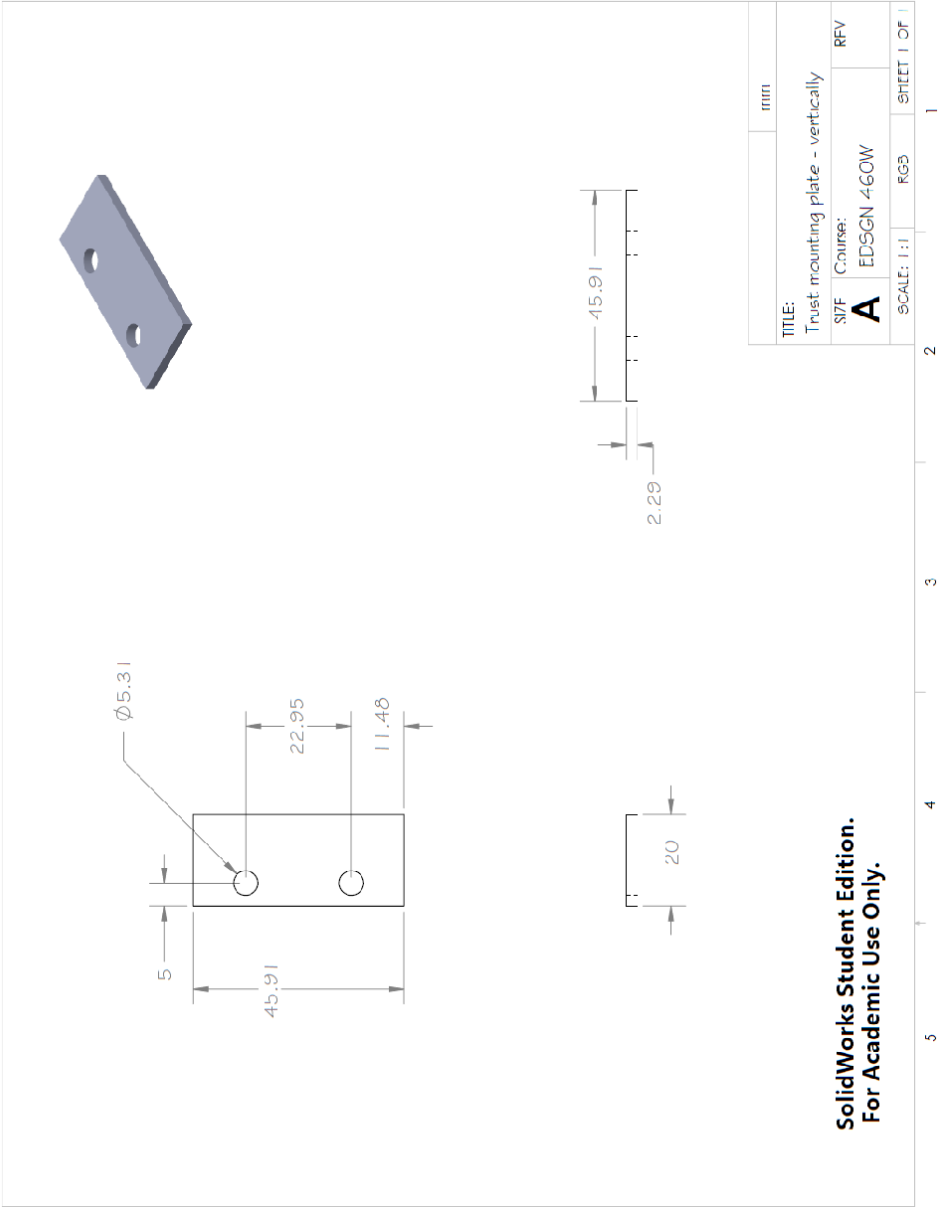
250HT2EA

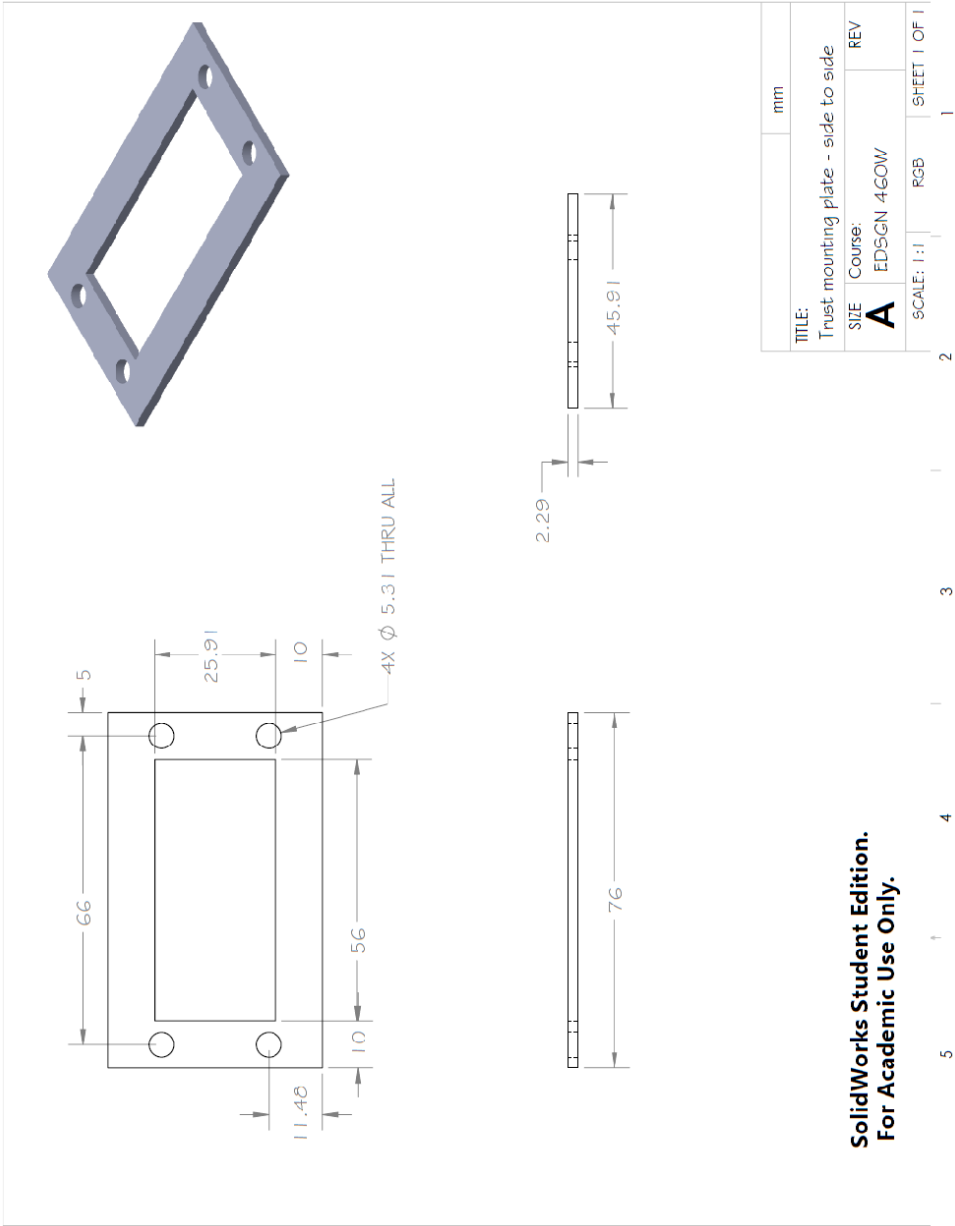
-  
Socket button head screw
 半圓頭內六角螺絲(0.4x4mm) x2
-  
Socket button head screw
 半圓頭內六角螺絲(0.4x5mm) x1
-  
Socket button head screw
 半圓頭內六角螺絲(0.4x8mm) x3
-  
Socket button head self tapping screw
 半圓頭內六角自攻螺絲(T1.5x6mm) x2

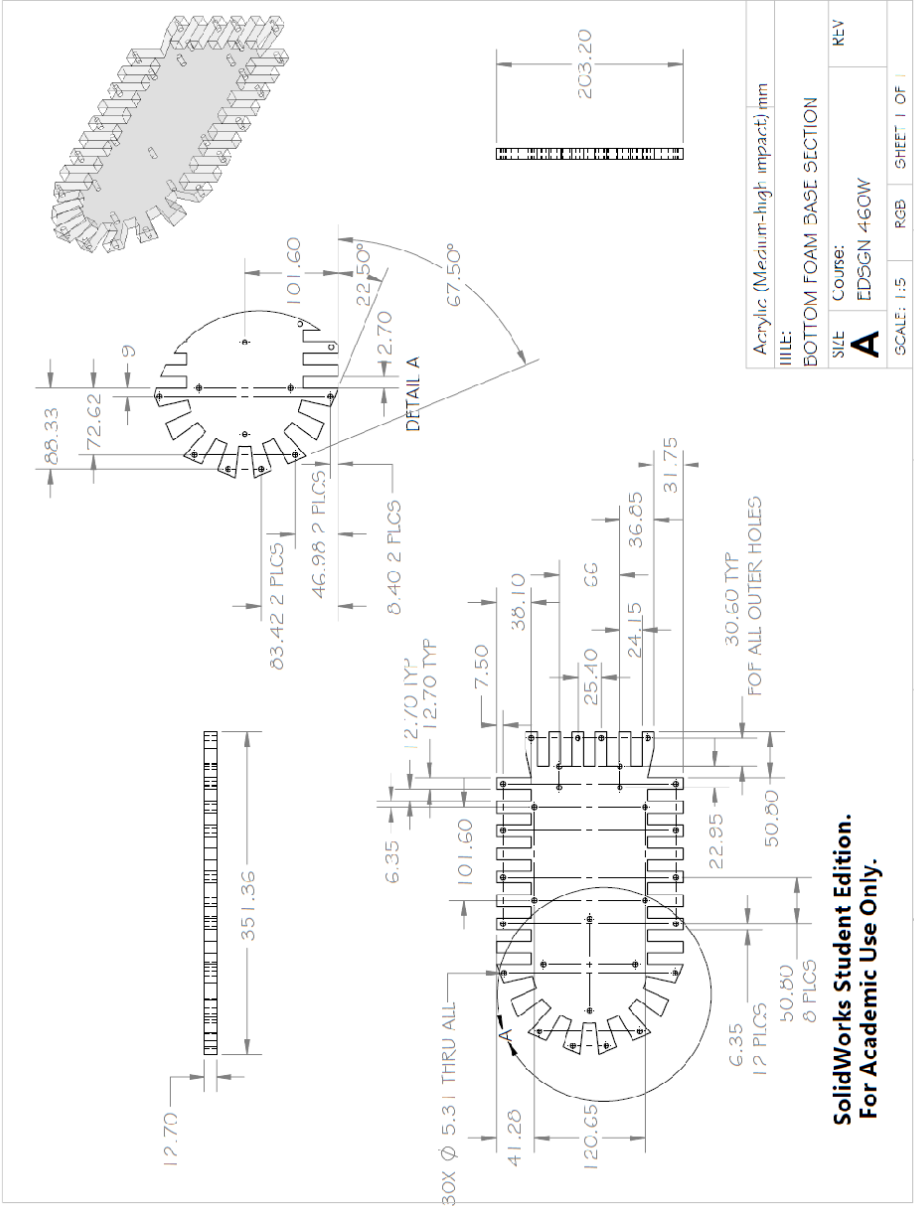


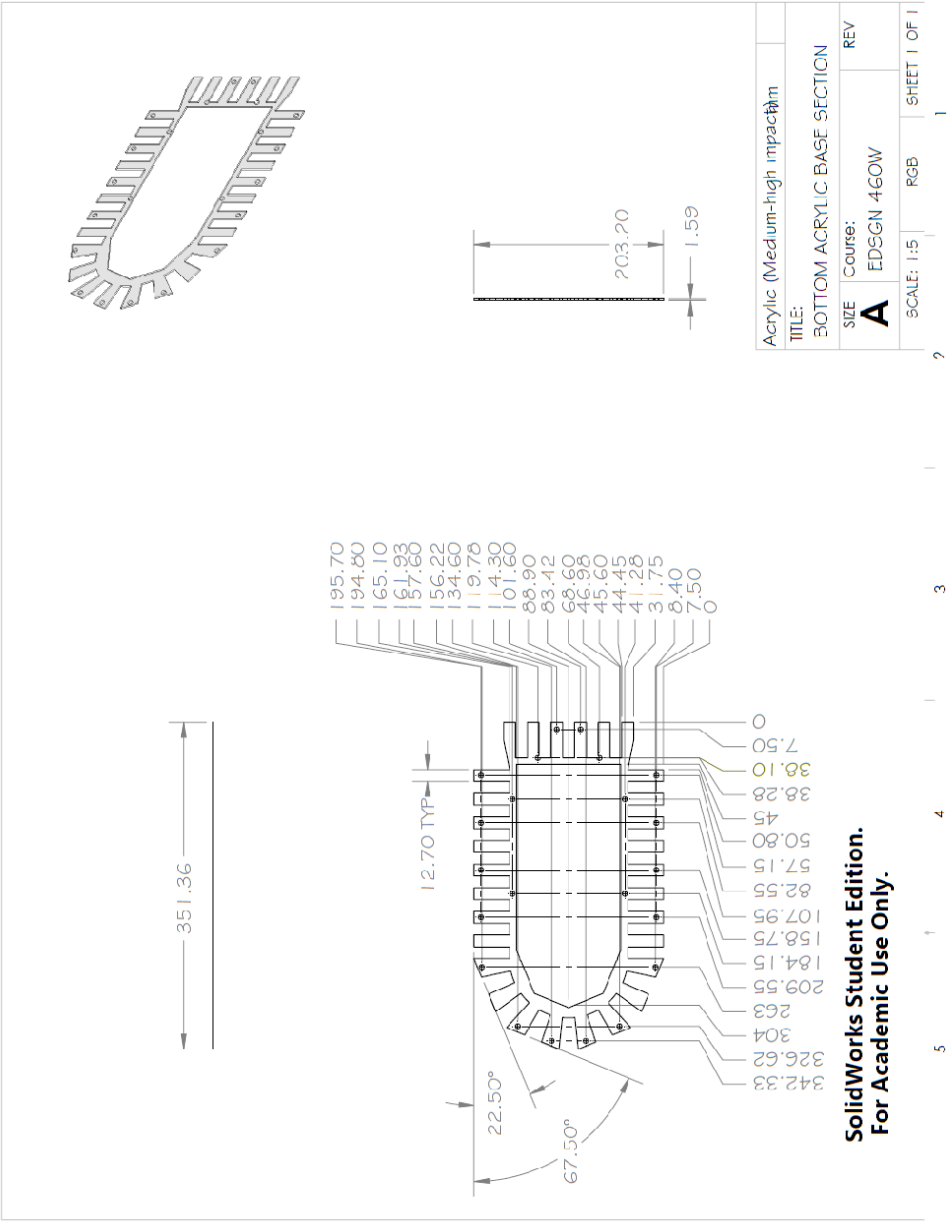
Appendix E: Detailed Computer Aided Design Drawings

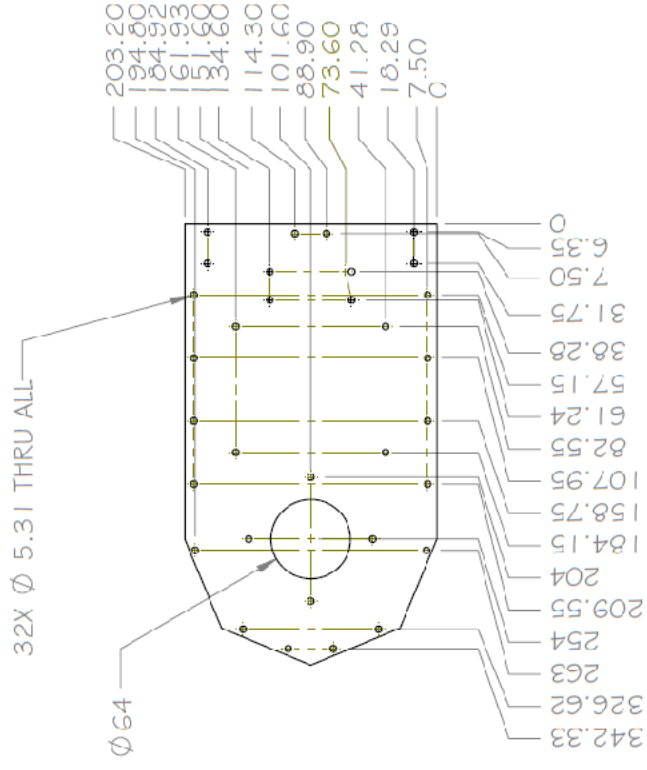
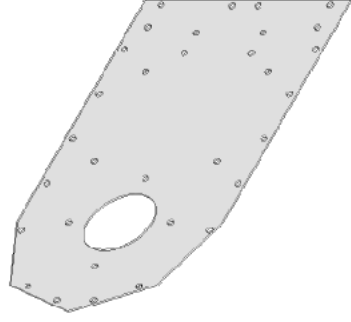
Manufactured Part Drawings



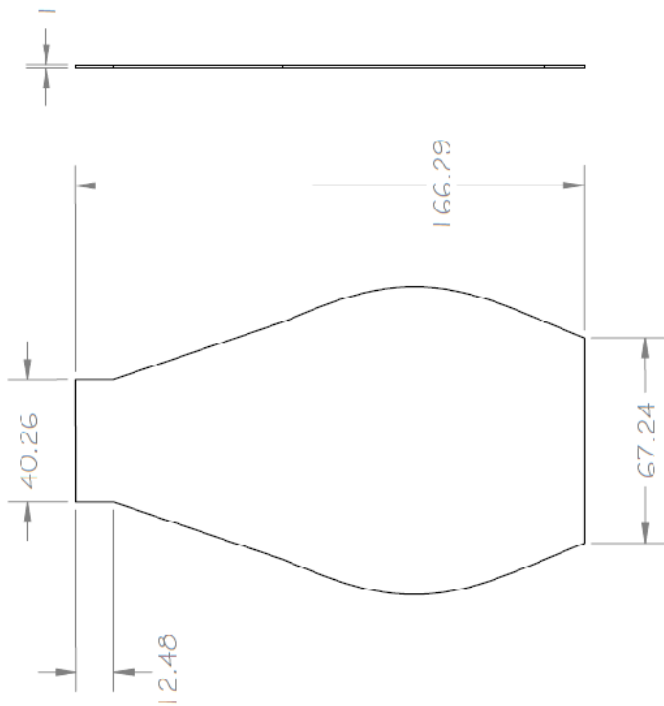






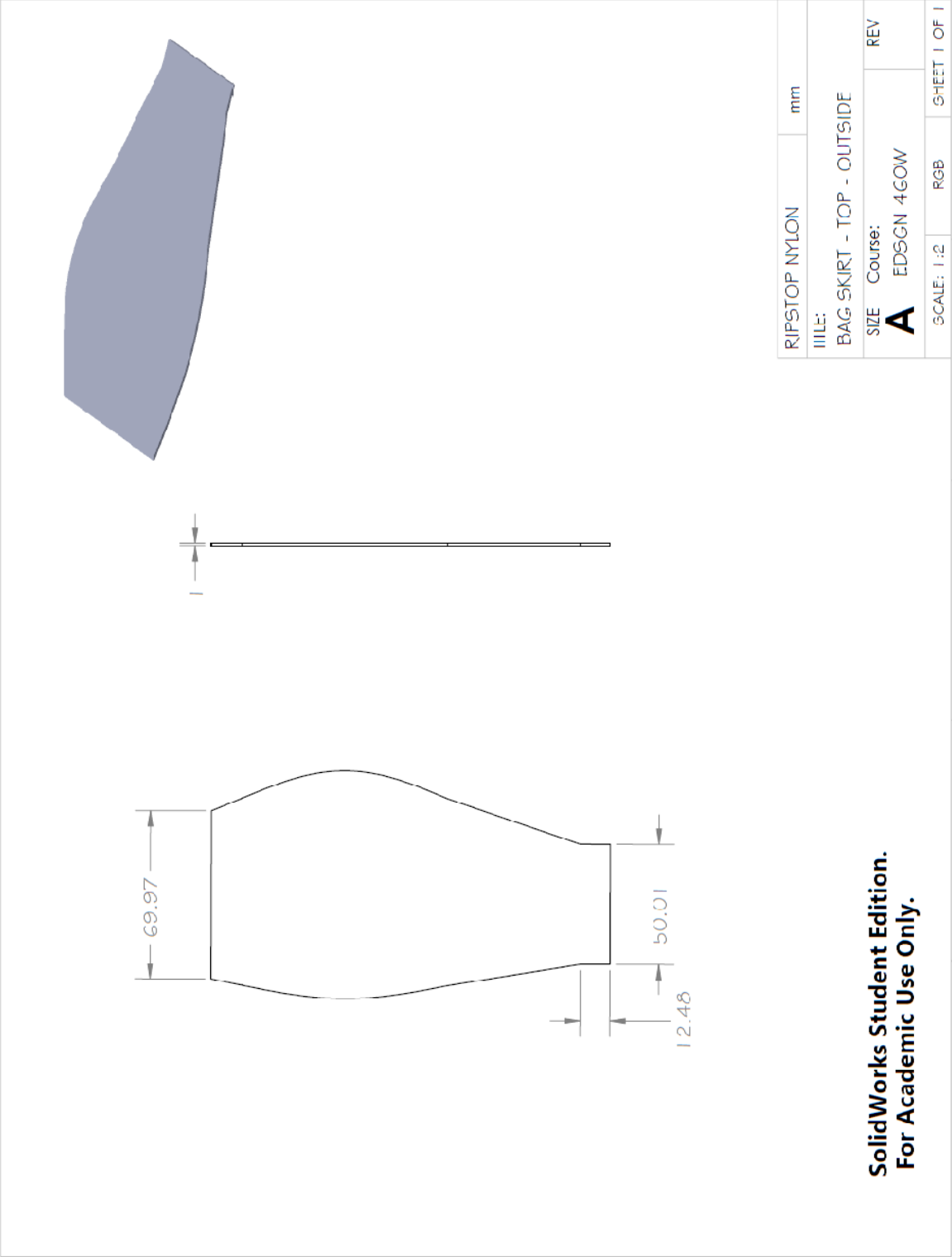


Acrylic (Medium-high impact)	
TITLE:	
Top and Bottom Base section	
SIZE	REV
A	EDSGN 4GOW
COURSE:	
SCALE: 1:5	RGB
SHEET 1 OF 1	

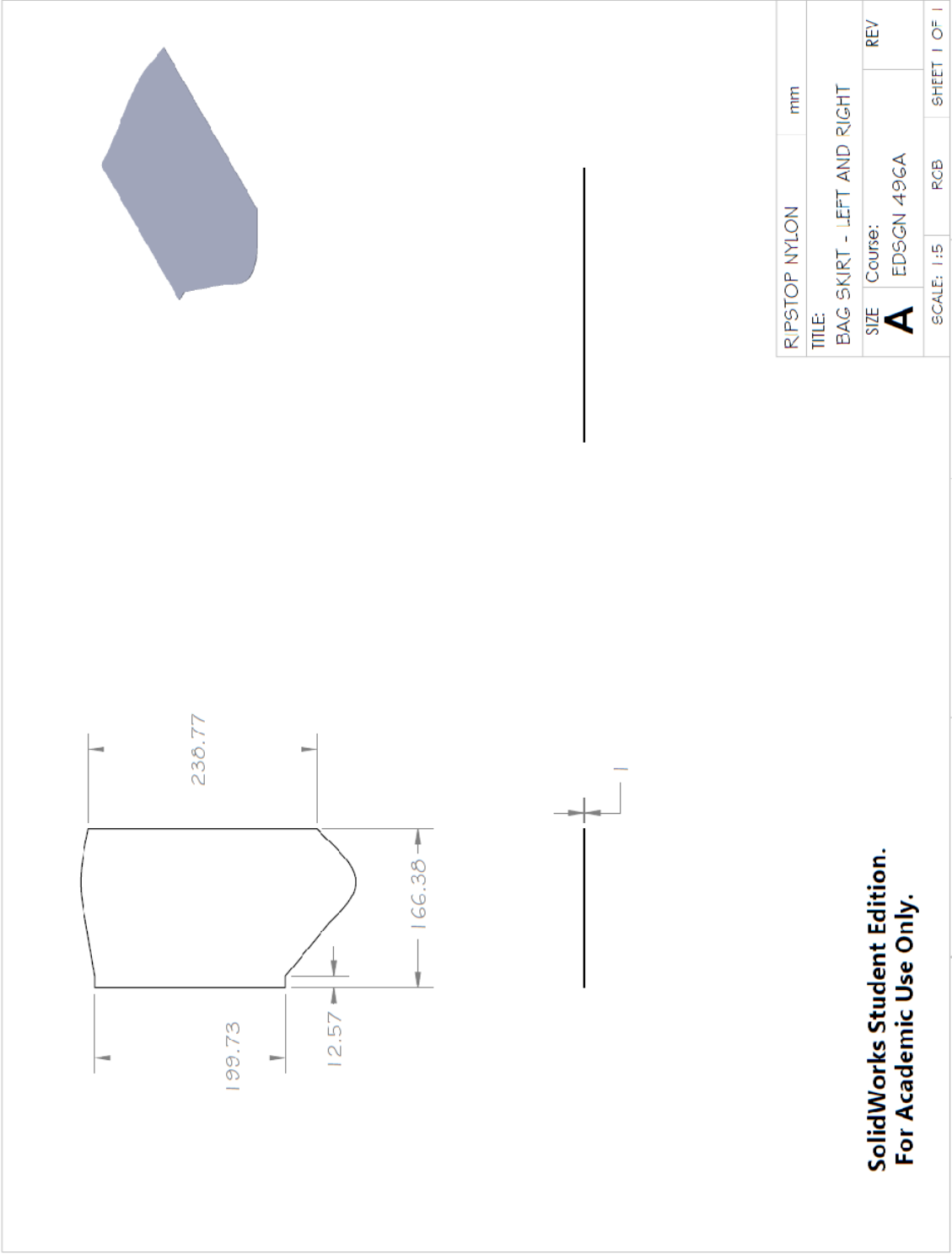


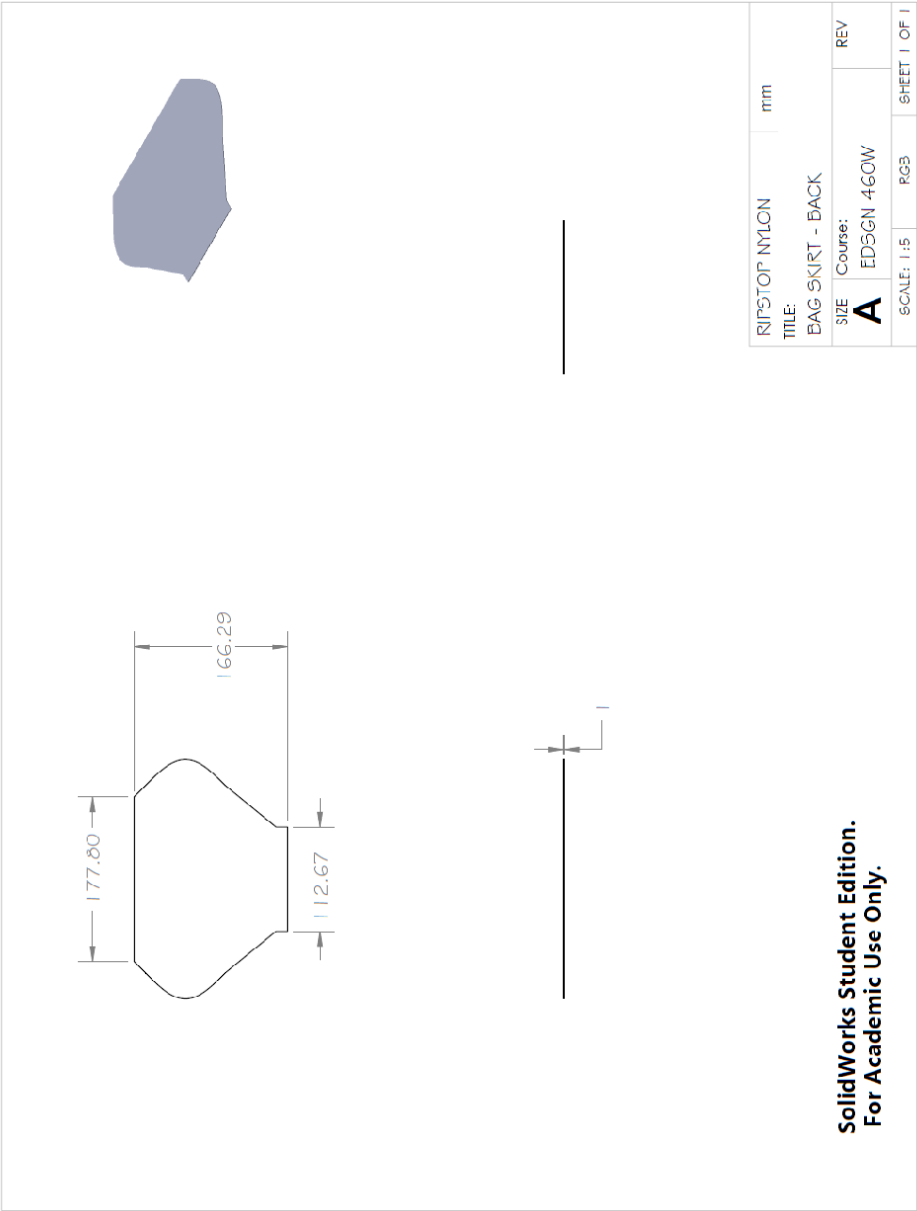
RIPSTOP NYLON		mm
TITLE: BAG SKIRT - TOP - CENTER		
SIZE A	Course: EDSGN 4GOW	REV
SCALE: 1:2		RGB
SHEET 1 OF 1		1

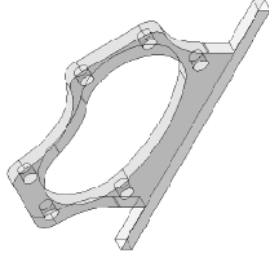
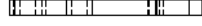
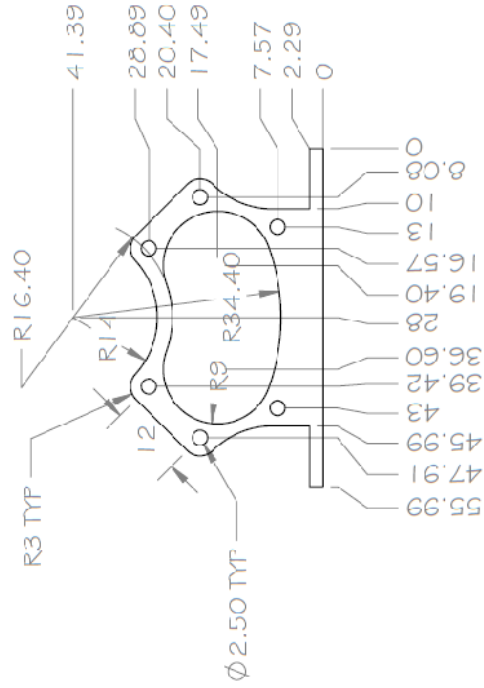
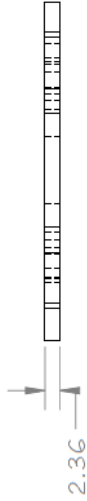
SolidWorks Student Edition.
For Academic Use Only.



SolidWorks Student Edition.
For Academic Use Only.

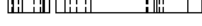
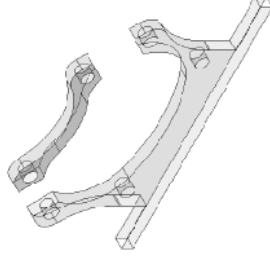
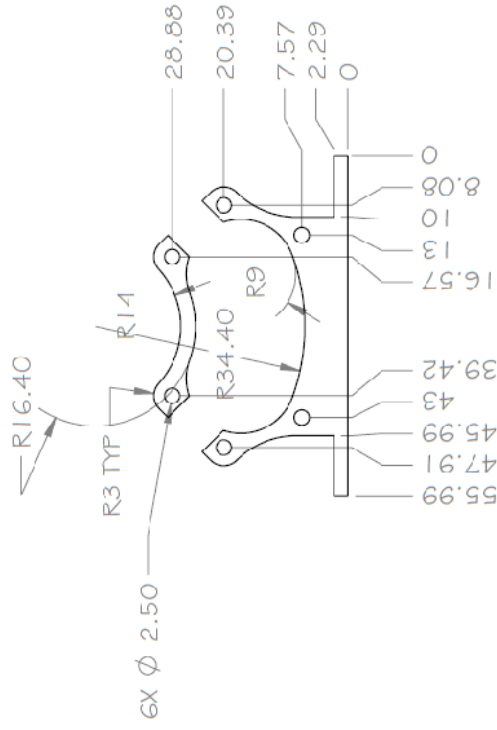
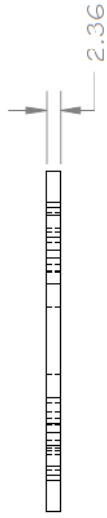






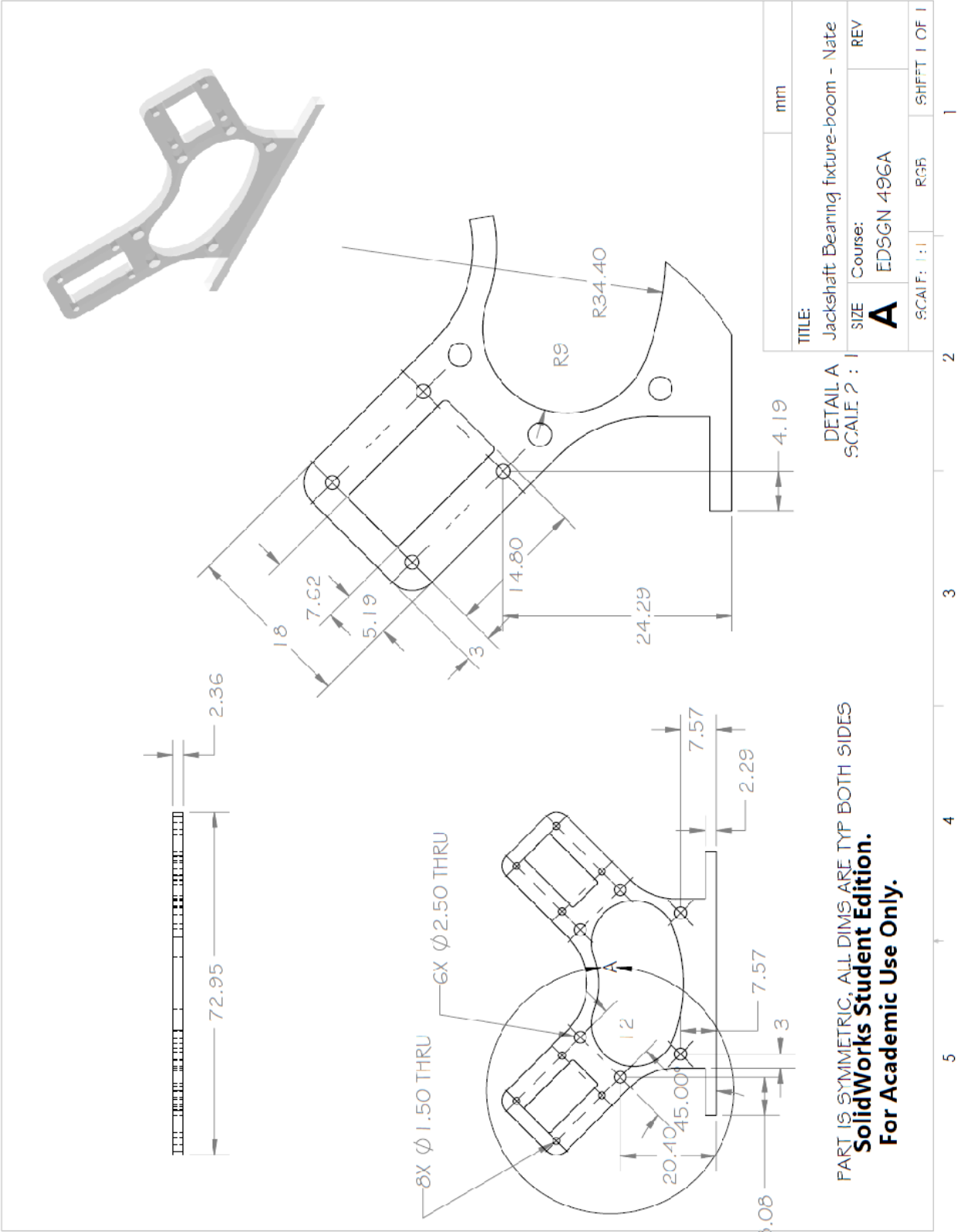
TITLE:		mm	
JACKSHAFT BEARING FIXTURE - REAR			
SIZE	Course:	REV	
A	EDSGN 4GOW		
SCALE: 1:1		RCB	SHEET 1 OF 1

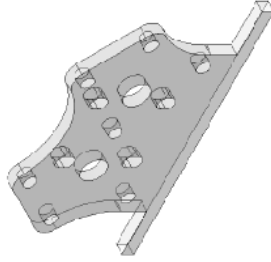
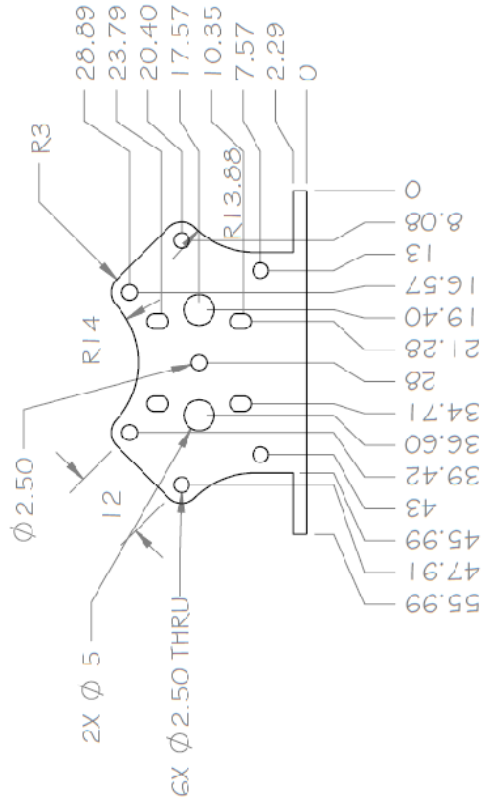
**SolidWorks Student Edition.
For Academic Use Only.**



ACRYLIC		mm
TITLE:		
JACKSHAFT BEARING FIXTURE - CASE		
SIZE	Course:	REV
A	EDSGN 460W	
SCALE: 1:1		RGB
SHEET 1 OF 1		

**SolidWorks Student Edition.
For Academic Use Only.**

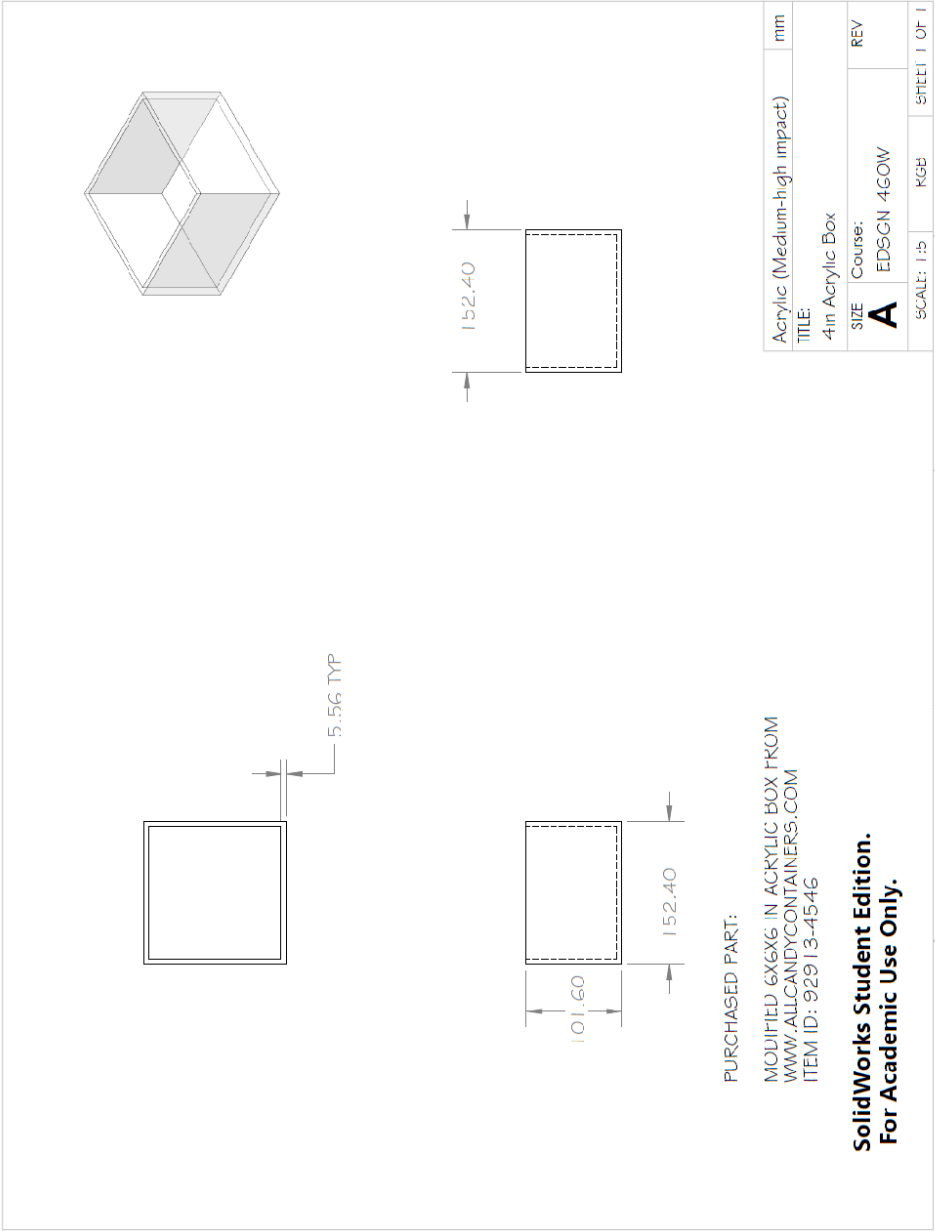


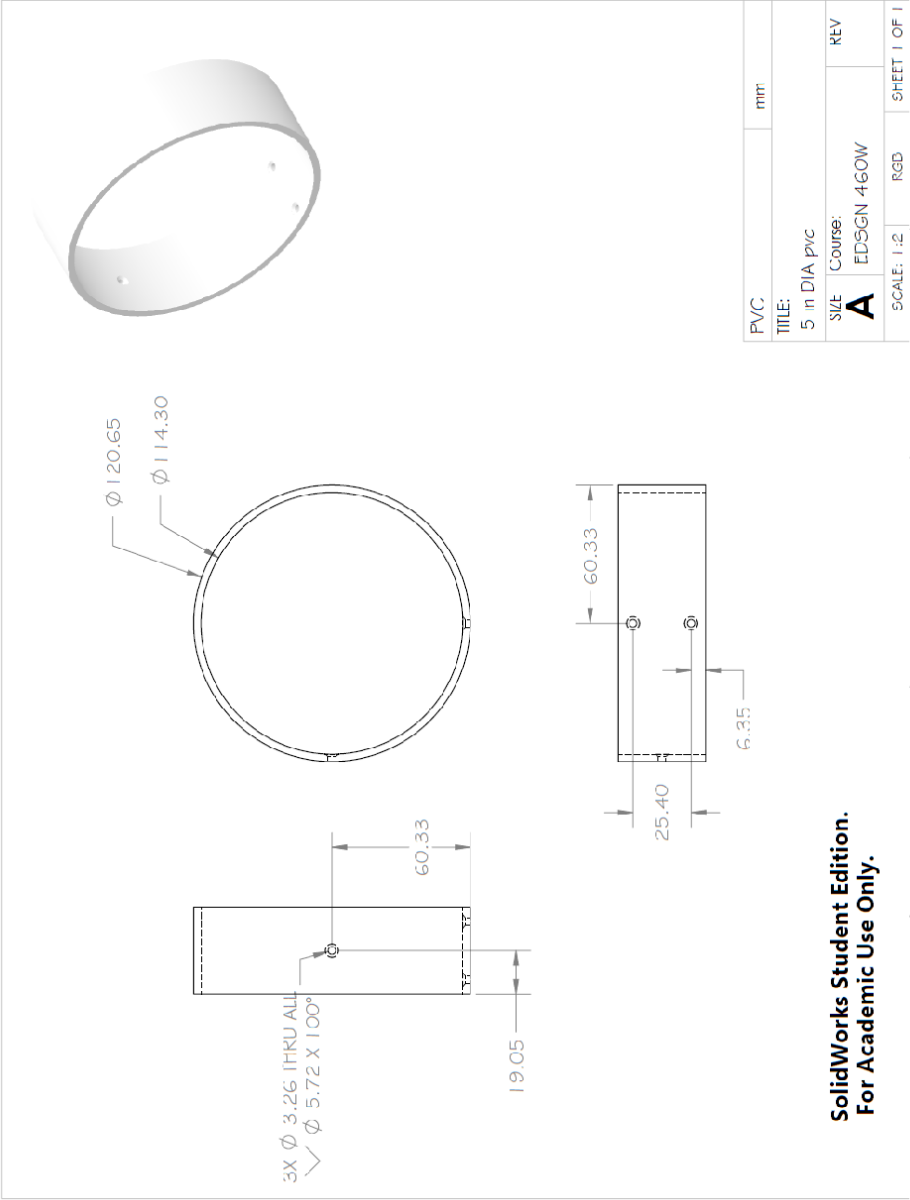


ACRYLIC		mm
TITLE: JACKSHAFT BEARING FIXTURE		
SIZE	Course: A	REV
EDSGN 460W		
SCALE: 1:1	RGB	SHEET 1 OF 1

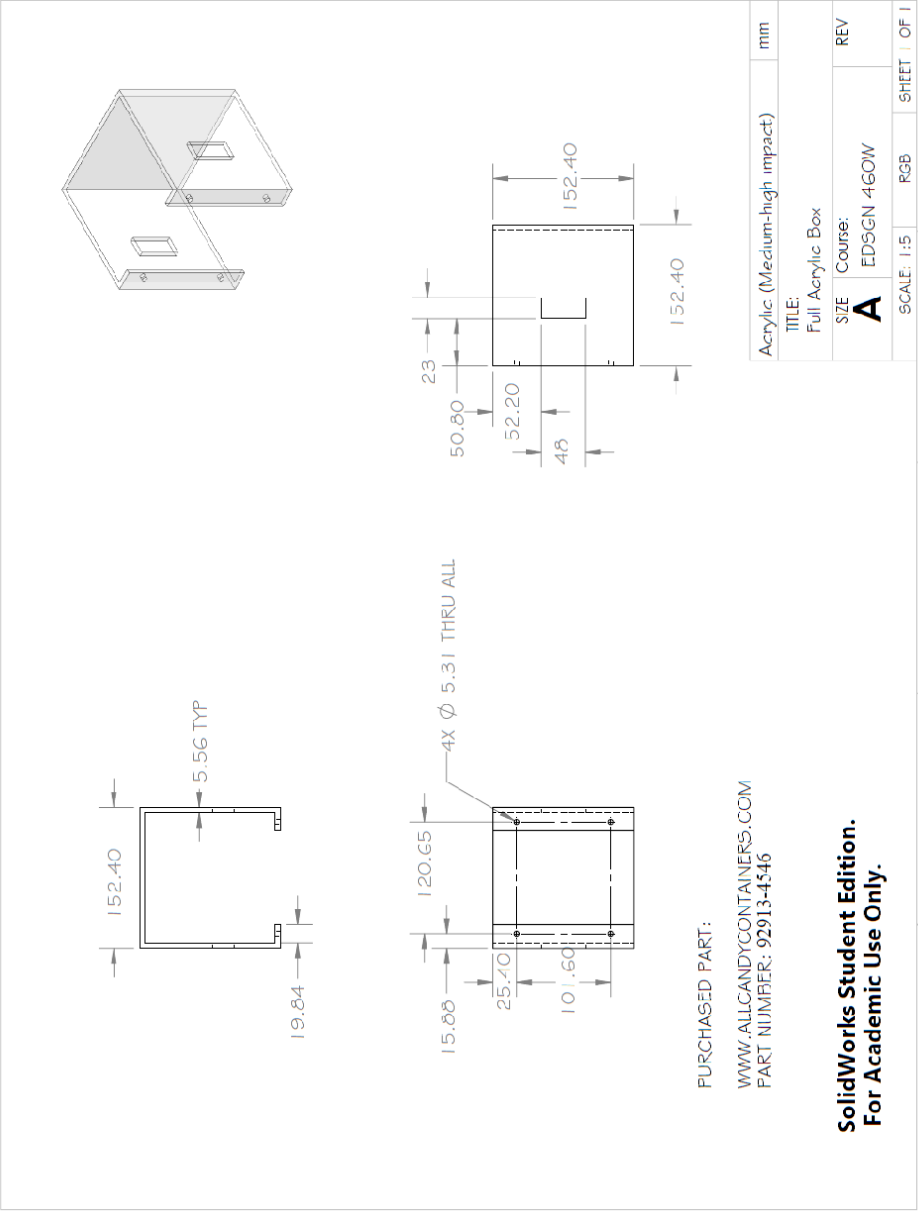
**SolidWorks Student Edition.
For Academic Use Only.**

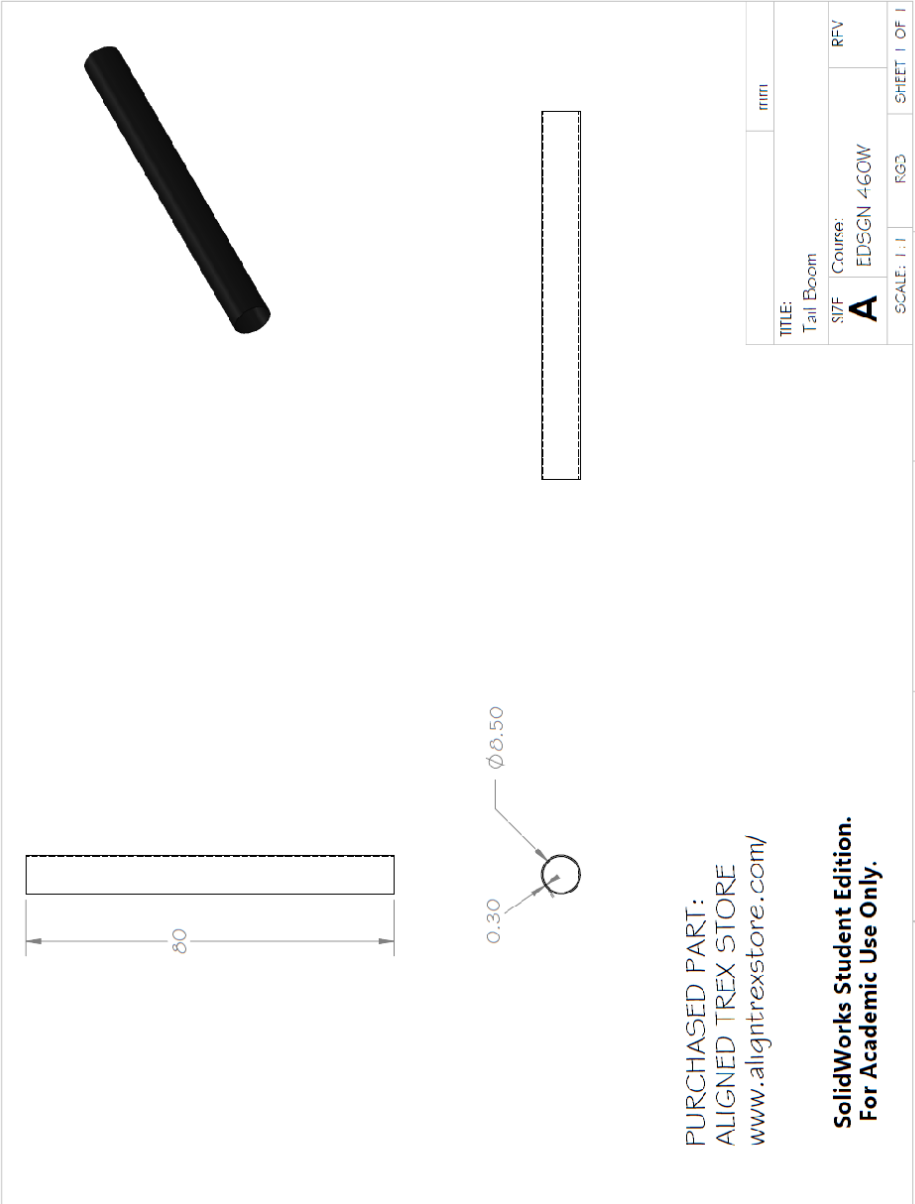
Modified Purchased Parts:





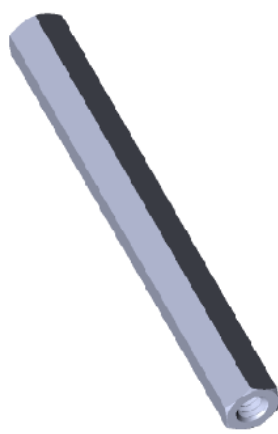
SolidWorks Student Edition.
For Academic Use Only.







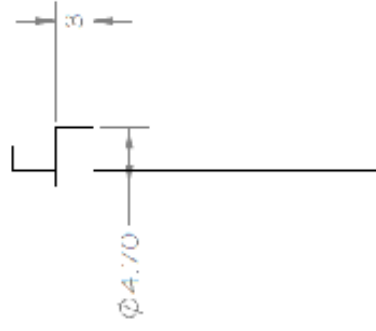
44.02



PURCHASED PART:
1 3/4 IN FEMALE ALUMINUM STANDOFF
SIZE 4

FROM MCMASTER-CARR
PART NUMBER 91760A124

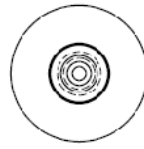
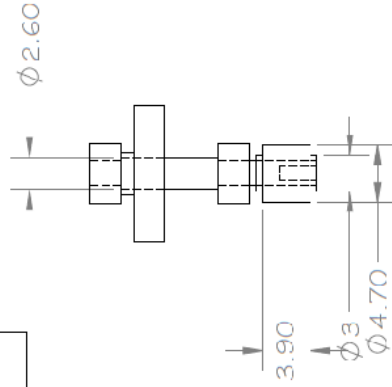
TITLE:		mm
1.75in aluminum standoff		
SIZE	COURSE:	REV
A	ED3GN 460W	
SCALE: 2:1	RGB	SHEET 1 OF 1



**SolidWorks Student Edition.
For Academic Use Only.**

UNIT: mm		ball belt unit - shaft	
SIL	Course:	EDS GN 46CW	NLV
SCALE: 2:1		80%	SHEET 1 OF 1

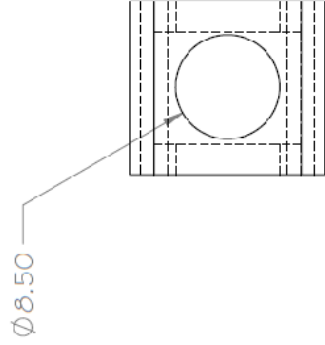
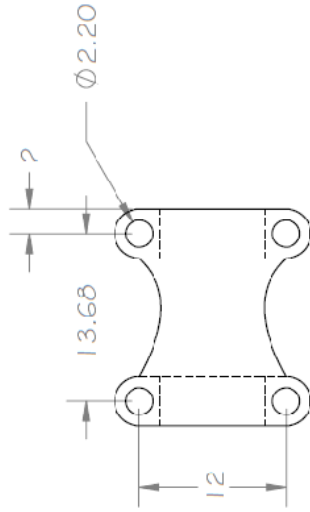
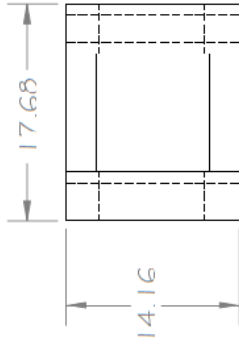
ITEM NO.	PART NUMBER	QTY.
1	Drive Pulley	1
2	Jackshaft	1
3	Jackshaft bearing	2
4	Jackshaft Gear	1



PURCHASED PART:
 ALIGNED TREX STORE
www.alignedtrexstore.com/

TITLE:		mm
Tail Drive Gear Assembly		
SIZE	COURSE:	REV
A	EDSGN 19CA	
SCALE: 2:1		SHEET 1 OF 1

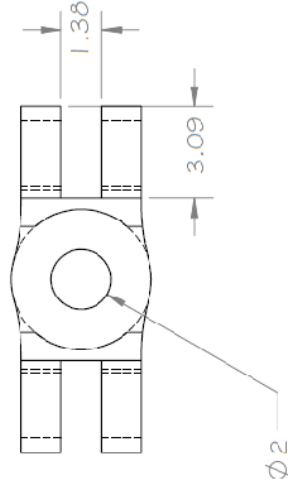
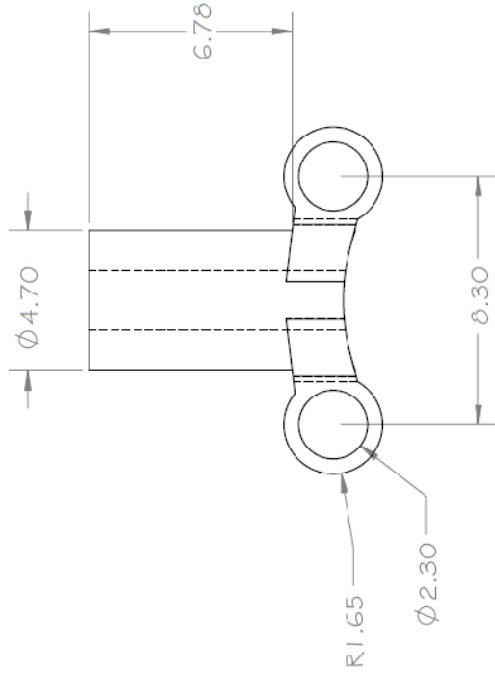
SolidWorks Student Edition.
For Academic Use Only.



PURCHASED PART:
 ALIGNED TREX STORE
www.aligntrexstore.com/

TITLE:		mm	
Tail Boom Mount			
SIZE	COURSE:	REV	
A	EDSGN 496A		
SCALE: 2:1		RGB	SHEET 1 OF 1

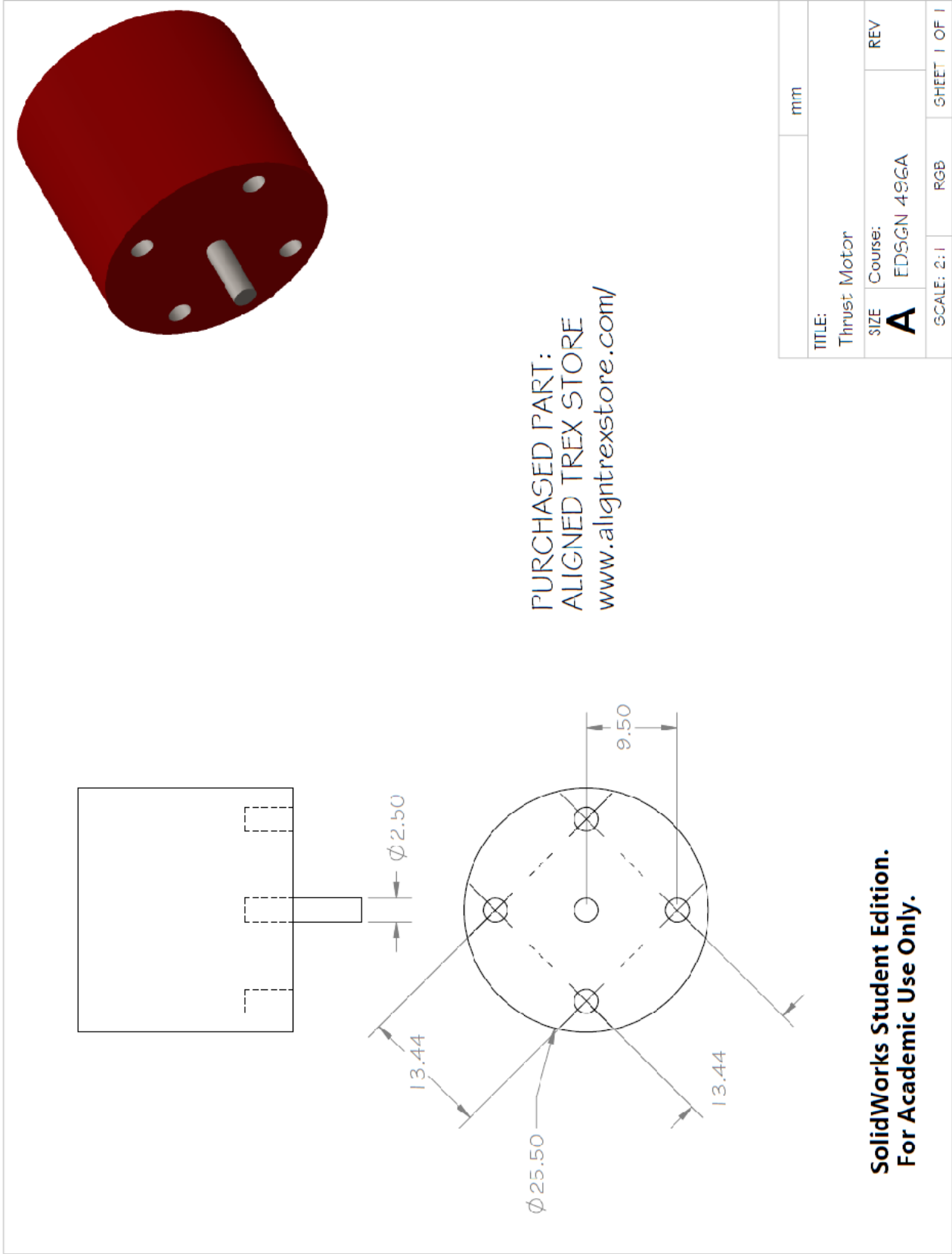
SolidWorks Student Edition.
For Academic Use Only.



PURCHASED PART:
 PURCHASED FROM WWW.ALIGNTREXSTORE.COM
 FROM THE TAIL PITCH ASSEMBLY OF A TREX 250
 MICRO HELICOPTER

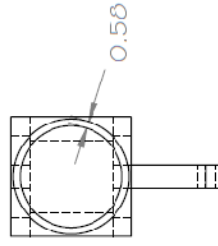
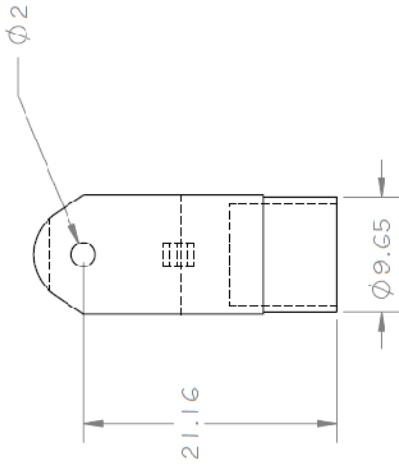
**SolidWorks Student Edition.
 For Academic Use Only.**

TITLE:		mm	
PROPELLER SHAFT BEARING			
SIZE	Course:	REV	
A	EDSGN 460W		
SCALE: 5:1	RGB	SHEET 1	OF 1



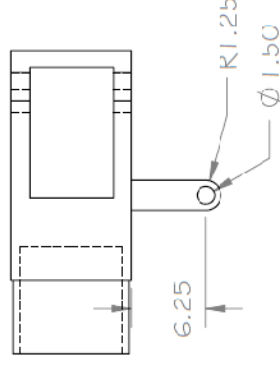
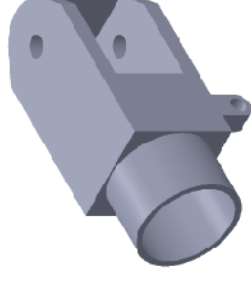
PURCHASED PART:
ALIGNED TREX STORE
www.aligntrexstore.com/

**SolidWorks Student Edition.
For Academic Use Only.**



PURCHASED PART:

FROM WWW.ALIGNTREXSTORE.COM

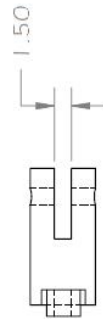


TITLE:		mm	
tail belt unit - body			
SIZE	COURSE:	REV	
A	EDSGN 49GA		
SCALE: 2:1	RGB	SHEET 1 OF 1	

**SolidWorks Student Edition.
For Academic Use Only.**

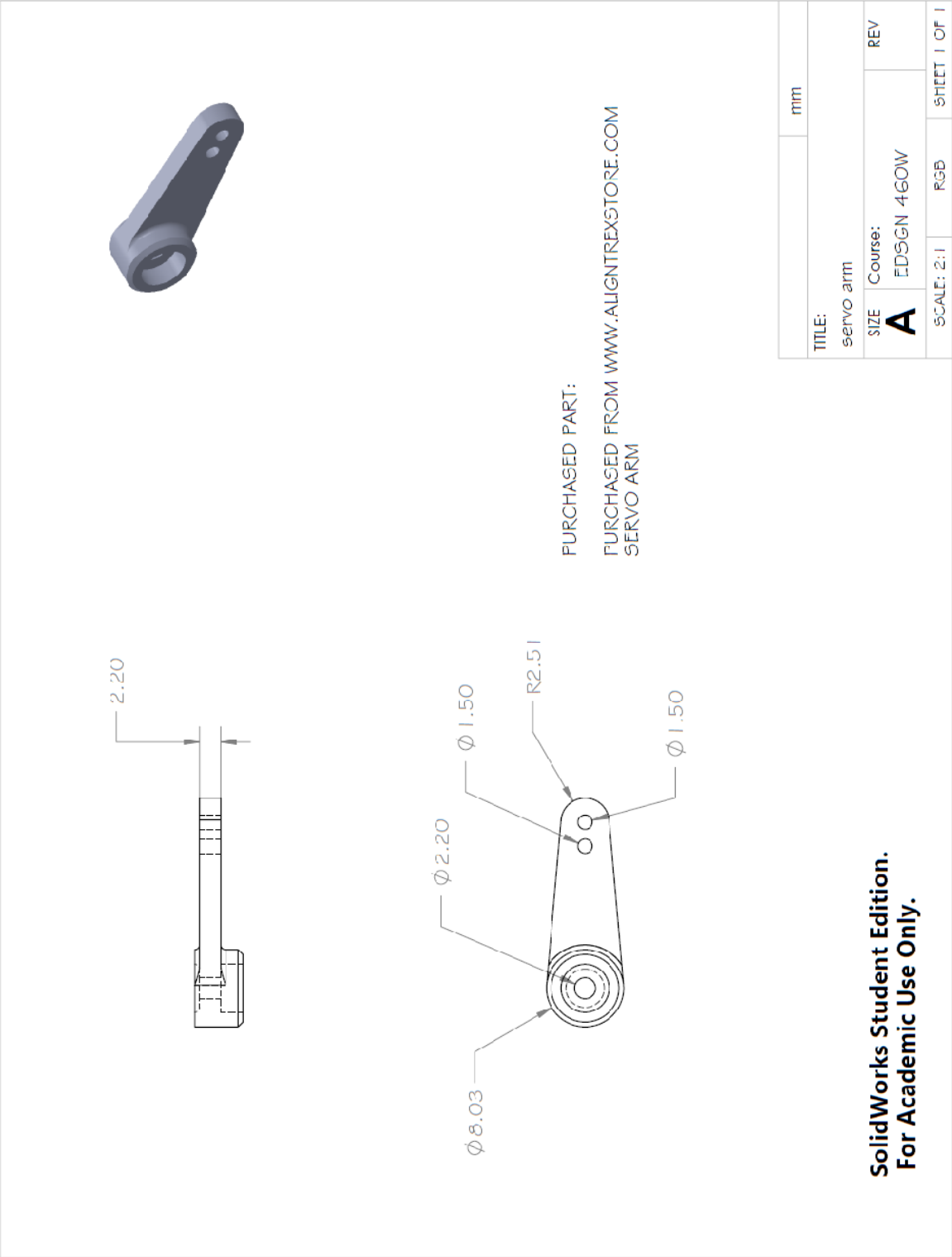


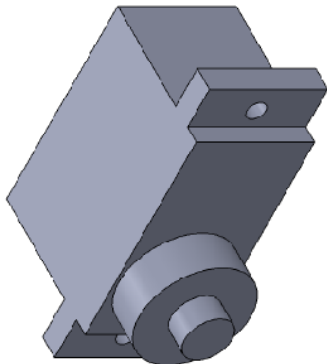
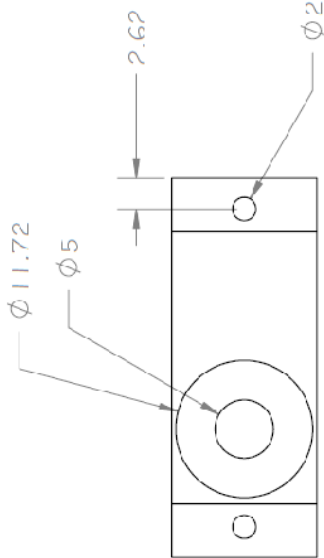
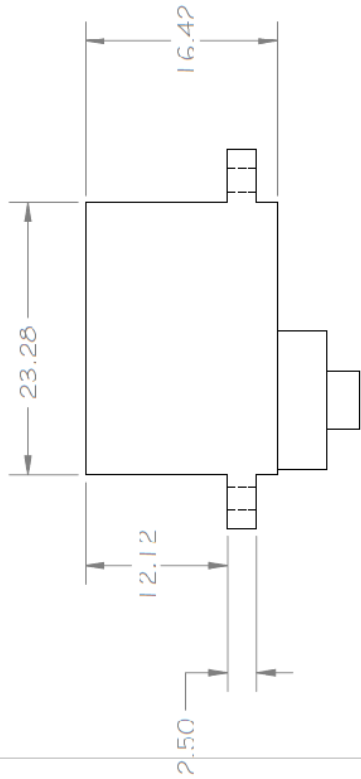
PURCHASED PART:
PURCHASED FROM WWW.ALIGNTREXSTORE.COM



**SolidWorks Student Edition.
For Academic Use Only.**

TITLE:		mm
Net Metal Tail Hole Set - outer plastic part		
SIZE	Course:	REV
A	EDSGN 460W	
SCALE: 2:1		RGB
2		SHEET 1 OF 1

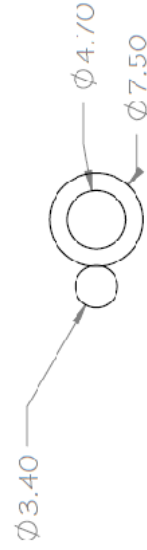
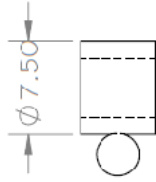




PURCHASED PART:
 PURCHASED FROM WWW.ALIGNTREXSTORE.COM
 PART NUMBER: HSD4160IT
 DS416M DIGITAL SERVO

SolidWorks Student Edition.
 For Academic Use Only.

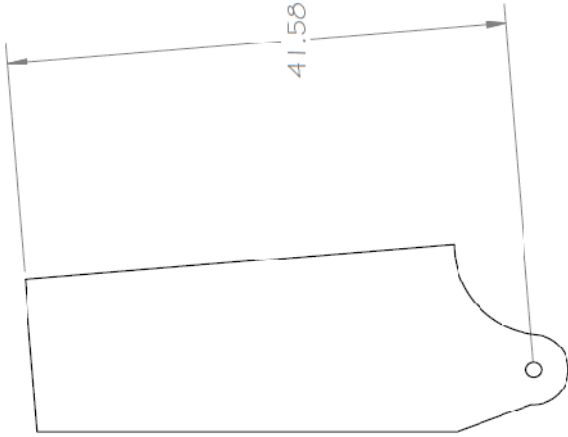
TITLE:		mm
VARIABLE PITCH PROPELLER SERVO		
SIZE	COURSE:	REV
A	EDSGN 4GOW	
SCALE:	2:1	RGB
SHEET 1 OF 1		1



PURCHASED PART:
PURCHASED FROM WWW.ALIGNTREXSTORE.COM
FROM THE TAIL PITCH ASSEMBLY

**SolidWorks Student Edition.
For Academic Use Only.**

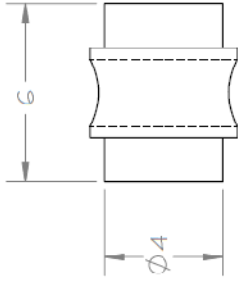
TITLE:		mm
prop collar		
SIZE	Course:	REV
A	EDSGN 4GOW	
SCALE: 2:1	RGB	SHEET 1 OF 1



PURCHASED PART:
PURCHASED FROM WWW.ALIGNTREXSTORE.COM

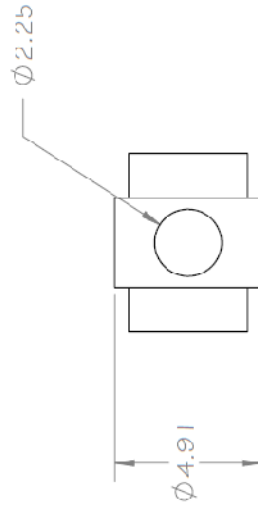
**SolidWorks Student Edition.
For Academic Use Only.**

TITLE: Prop		mm	
SIZE A	Course: EDSGN 4GOW	REV	
SCALE: 2:1		RGD	SHEET 1 OF 1



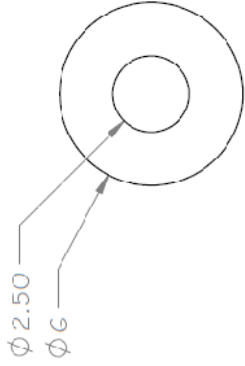
PURCHASED PART:

PURCHASED FROM WWW.ALIGNTREXSTORE.COM



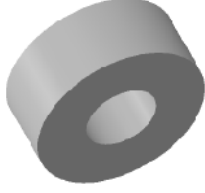
**SolidWorks Student Edition.
For Academic Use Only.**

TITLE:		mm	
New Metal Tail Holder set - Center piece			
SIZE	Course:	REV	
A	EDSGN 4GOW		
SCALE: 5:1		RGB	SHEET 1 OF 1



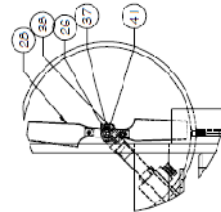
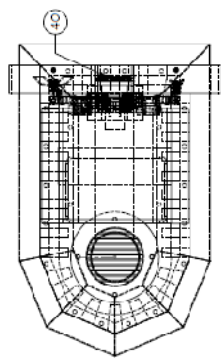
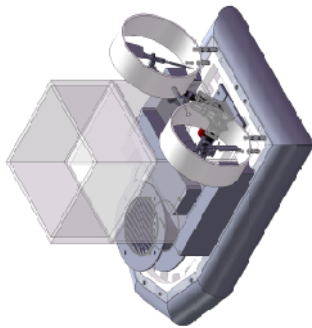
PURCHASED PART:
ALIGNED TREX STORE
www.alignedtrexstore.com/

SolidWorks Student Edition.
For Academic Use Only.

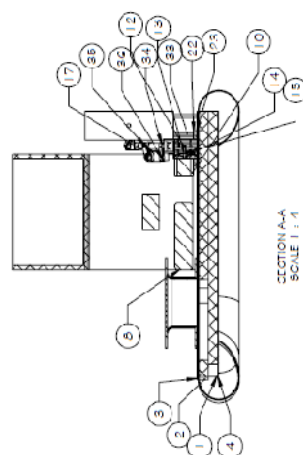
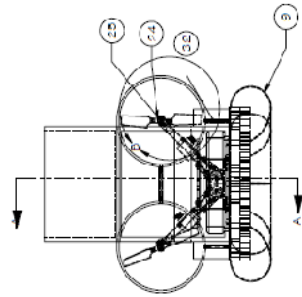
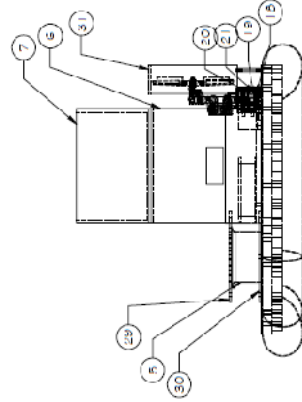


TITLE:		mm	
Motor pinion gear			
SIZE	Course:	REV	
A	EDSGN 4GOW		
SCALE: 5:1		RGB	SHEET 1 OF 1

ITEM NO.	PART NUMBER	QTY.
1	Bottom Base section	1
2	Middle Base Section	1
3	Top and Bottom Base section	1
4	Bottom Base	1
5	Genesis 3 In lift fan	1
6	Full Acrylic Box	1
7	4in Acrylic Box	1
8	Myxo	1
9	Bag Shirt - Nats- sheetmetal	1
10	Trust Motor	1
11	Motor pinion gear	1
12	Drive Pulley	2
13	Jackshaft	2
14	Jackshaft bearing	4
15	Jackshaft Gear	2
16	Tail Boom Mount	2
17	Tail Boom	2
18	Jackshaft Bearing fixture - Nats	2
19	Jackshaft Bearing fixture - case - Nats	3
20	Jackshaft Bearing fixture - Nats - boom	4
21	Jackshaft Bearing fixture - boom - Nats	2
22	Tail mounting plate - vertically	1
23	Tail mounting plate - size to side	1
24	Tail belt unit - body	2
25	Tail belt unit - shaft	2
26	Net Metal Tail Holder Set - outer plastic part	4
27	New Metal Tail Holder set - Center piece	2
28	Prop	4
29	Lift fan mounting piece top	1
30	Lift fan mounting piece bottom	1
31	5 in DIA. pvc	2
32	batteries	3
33	91780A034	4
34	clamp	4
35	servo arm	2
36	servo	2
37	prop shaft bearing	2
38	control arm	4
39	prop collar	2
40	91780A038	1
41	control link	2



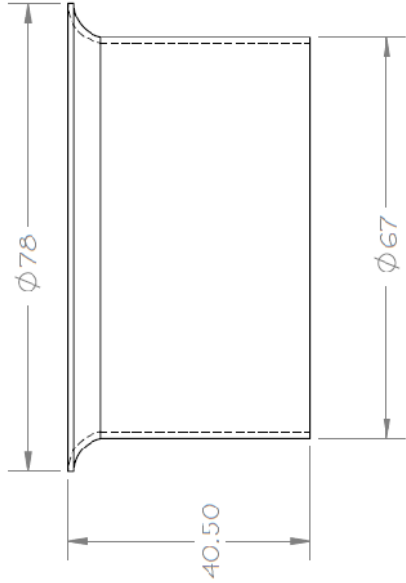
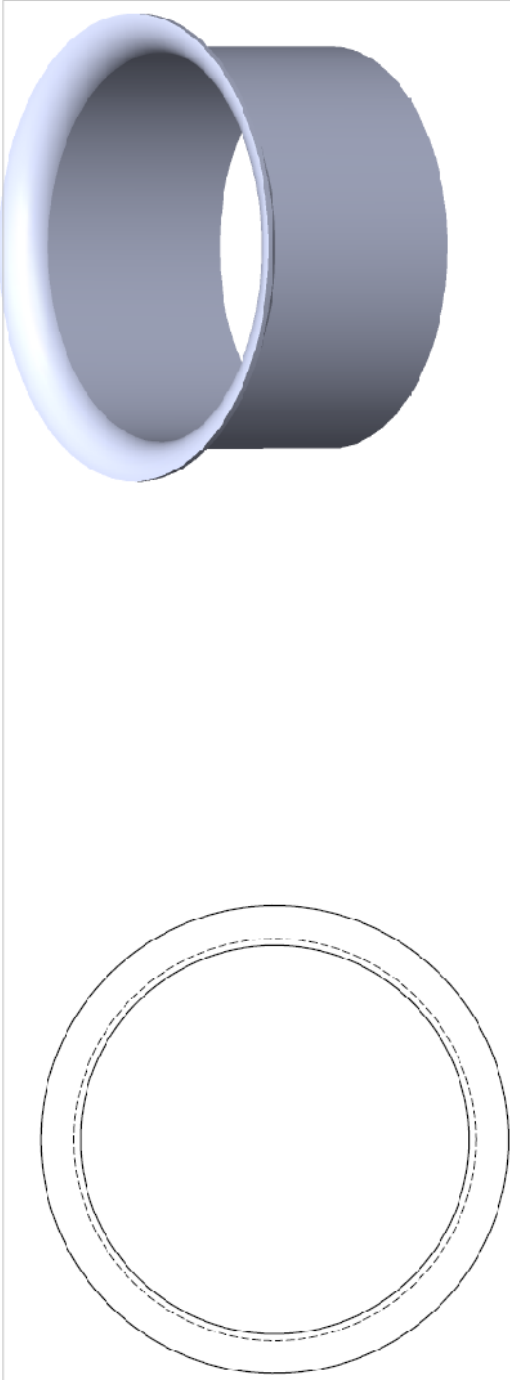
DETAIL B
SCALE 1 : 2



SECTION A-A
SCALE 1 : 4

TITLE:		mm	
Main Hovercraft Assembly			
SIZE	Course	REV	
A	EDSON 400W		
SCALE 1 : 2	WGS	5 SET	1 OF 1

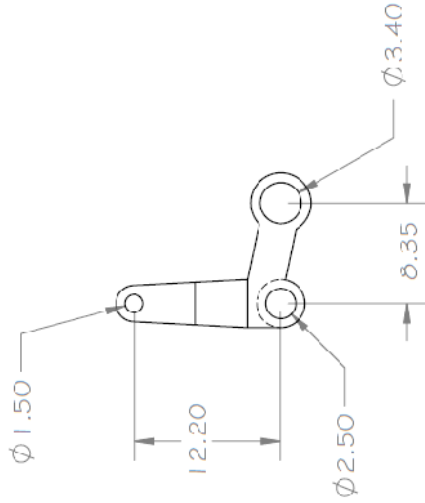
SolidWorks Student Edition.
For Academic Use Only.



EDF 64MM DUCTED FAN WITH 4750KV MOTOR
PPURCHASED FROM AMAZON.COM

TITLE: 64MM EDF		mm	
SIZE	COURSE:	REV	
A	EDSGN 460W		
SCALE: 1:1	RGB	SHEET 1 OF 1	

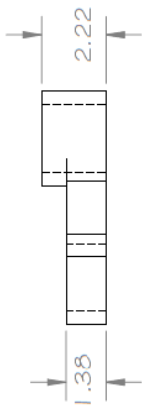
SolidWorks Student Edition.
For Academic Use Only.



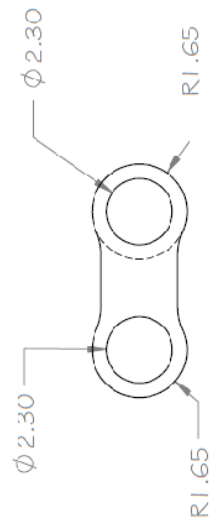
PURCHASED PART:
PURCHASED FROM ALIGNREXSTORE.COM
CONTROL LINK

TITLE:		mm
CONTROL LINK		
SIZE	Course:	REV
A	EDSGN 460W	
SCALE: 2:1	RGB	SHEET 1 OF 1

SolidWorks Student Edition.
For Academic Use Only.



PURCHASED PART:
PURCHASED FROM WWW.ALIGNREXSTORE.COM



**SolidWorks Student Edition.
For Academic Use Only.**

TITLE:		mm	
control arm			
SIZE	Course:	REV	
A	EDSGN 460W		
SCALE: 5:1	RGB	SHIFT 1 OF 1	1

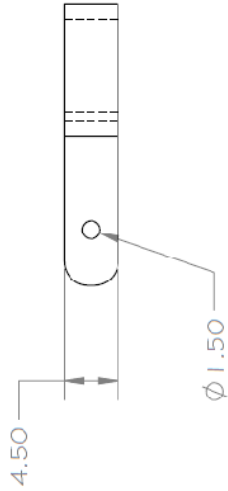
2

3

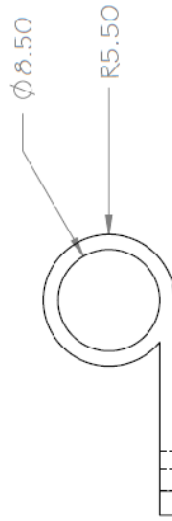
4

5

1

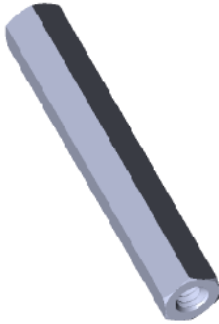
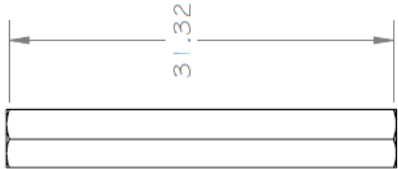


PURCHASED PART:
PURCHASED FROM WWW.ALIGNTREXSTORE.COM
RUDDER SERVO MOUNT



**SolidWorks Student Edition.
For Academic Use Only.**

TITLE:		mm
RUDDER SERVO MOUNT		
SIZE	Course:	REV
A	EDSGN 460W	
SCALE: 2:1	RGB	SHEET 1 OF 1



PURCHASED PART:

PURCHASED FROM MCMASTER CARR
1.25IN, 4-40 ALUMINUM STANDOFF
PART NUMBER: 91780A118

**SolidWorks Student Edition.
For Academic Use Only.**

ALUMINIUM		mm	
TITLE: 1 1/4 IN Aluminum Female Threaded Hex			
SIZE A	COURSE: EDSGN 460W	REV	
SCALE: 2:1		RG3	SHEET 1 OF 1
2	1		

5

4

3

2

1

Appendix F: Complete Bill of Materials

	Qty	PN	Description	Vendor	Unit Cost (\$)	Shipping (\$)	Subtotal (\$)	
Thrust Components	2	H25095A	Metal Tail Holder Set	Align T-rex Store	14.99	0	29.98	
	2	H25021	Tail Pitch Assembly	Align T-rex Store	9.99	0	19.98	
	2	H25026B	Metal Tail Belt Unit	Align T-rex Store	24.99	0	49.98	
	2	H25062-1	Tail Rotor Control Arm Set	Align T-rex Store	4.99	0	9.98	
	1	H25030-00	Tail Boom	Align T-rex Store	5.99	0	5.99	
	4	HQ0423A	42 Tail Blades	Align T-rex Store	2.99	0	11.96	
	2	H25023	Rudder Servo Mount	Align T-rex Store	2.99	0	5.98	
	2	H25020	Tail Boom Mount	Align T-rex Store	5.99	0	11.98	
	4	H25029	Tail Drive Gear Assembly	Align T-rex Store	5.99	0	23.96	
	2	H25048	Motor Pinion Gear	Align T-rex Store	4.99	0	9.98	
	2	HSD41601	DS416M Digital Servo	Align T-rex Store	24.99	0	49.98	
	1	HML25M01	250MX Brushless Motor	Align T-rex Store	34.99	0	34.99	
	4	A 6B18M230030	1 mm Pitch, 230 Teeth, 3mm wide Single Sided Polyurethane Belt with Kevlar Cords	SDP-SI	2.18	6.405	15.13	
	4	A 6B18M225030S	1 mm Pitch, 225 Teeth, 3mm wide Single Sided Polyurethane Belt with Kevlar Cords	SDP-SI	1.45	6.405	12.21	
	Lift Components	2	HL6408 2431-5250KV	EDF Ducted Fan Power System	Amazon	23.67	0	47.34
	Electronic Components	1	EFLA1030B	30-Amp Pro Switch-Mode BEC Brushless ESC V2	HobbyTown USA	44.99	0	44.99
		3	NA	Turnigy 4000 mAh LiPo	Boeing, donated	0	0	0.00
1		NA	MyRio 1900	Boeing, donated	0	0	0.00	
1		40067135	NEEWER® RC Helicopter 40A ESC Brushless Motor Speed Controller BEC 3A	Amazon	12.25	3.99	16.24	
1		SPMAR400	Spektrum AR400 AR400 4-Channel DSMX Aircraft Receiver	Amazon	29.99	0	29.99	
1		40070454	NEEWER® 3.5mm Gold Plated Bullet Banana Plug Connector for RC Battery 10-Pairs	Amazon	6.95	3.99	10.94	
Materials	2	92913-4546	Square Acrylic Display Bin	Candy Containers	8.99	12.7	30.68	
	1	754404	4x8' Rigid Foam	Home Depot	12.76	0	12.76	
	2	241610	20"x32"x0.093" Acrylic Sheet	Home Depot	14.98	0	29.96	
	1	429-040	4 in. Schedule 40 PVC Coupling	Home Depot	3.92	0	3.92	
	4	RIPIBLA	Ripstop Nylon (per yard)	Online Fabric Store	5.10	6.95	27.35	
	2	M10185624	Foam Board	Michaels	6.99	0.00	13.98	
	1	470797	Trash Bags	Lowes	7.98	0.00	7.98	
Hardware	4	91780A118	Aluminum Female Threaded Hex Standoff, 3/16" Hex, 1-1/4" Length, 4-40 Screw Size	McMaster-Carr	0.56	1.79	4.03	
	1	9273K14	General Purpose Nylon Hook and Loop, 1" Width X 5' Length, Adhesive Back, Black, Lengths of 5	McMaster-Carr	7.94	1.79	9.73	
	1	91780A124	Aluminum Female Threaded Hex Standoff, 3/16" Hex, 1-3/4" Length, 4-40 Screw Size	McMaster-Carr	0.62	1.79	2.41	
	1	90273A110	Zinc-Plated Steel Flat Head Phillips Machine Screw, 4-40 Thread, 1/2" Length, Packs of 100	McMaster-Carr	3.81	1.79	5.60	
	1	90272A119	Zinc-Plated Steel Pan Head Phillips Machine Screw, 4-40 Thread, 1-1/2" Length, Packs of 50	McMaster-Carr	5.22	1.12	6.34	
	1	90272A108	Zinc-Plated Steel Pan Head Phillips Machine Screw, 4-40 Thread, 3/8" Length, Packs of 100	McMaster-Carr	1.49	1.12	2.61	
	1	90272A110	Zinc-Plated Steel Pan Head Phillips Machine Screw, 4-40 Thread, 1/2" Length, Packs of 100	McMaster-Carr	1.63	1.12	2.75	
	1	90760A005	Zinc-Plated Steel Undersized Machine Screw Hex Nut, 4-40 Thread Size, 3/16" Width, 1/16" Height, Packs of 100	McMaster-Carr	2.36	1.12	3.48	
	1	90480A005	Zinc-Plated Steel Machine Screw Hex Nut, 4-40 Thread Size, 1/4" Width, 3/32" Height, Packs of 100	McMaster-Carr	0.81	1.12	1.93	
	1	90126A505	Zinc-Plated Steel SAE Flat Washer, Number 4 Screw Size, 0.125" ID, 0.312" OD, Packs of 100	McMaster-Carr	1.37	1.12	2.49	
	10	92510A294	Aluminum Unifthreaded Spacer, 3/16" OD, 1/8" Length, #4 Screw Size	McMaster-Carr	0.28	1.12	3.92	
	Poster	1	NA	Show case Display Board	Copy Center	53.34	0.00	53.34
					Total	666.79		

Appendix G: Arduino Code RC Controls

<pre> #include <Servo.h> #include "PPMrcIn.h" #define NUM_AVERAGES 3 #define CENTER_PPM 1500 #define PIN_LEFT_ROTOR 5 #define PIN_RIGHT_ROTOR 6 #define PIN_THROT_OUT 3 #define PIN_LIFT_OUT 12 #define PIN_X_IN 4 #define PIN_Y_IN 7 #define PIN_THROT_IN 2 // Channels read the readings from the reciever. Channel XIn; Channel YIn; Channel ThrotIn; // We use the Servo structure to control everything (including motors). Servo LeftRotor; Servo RightRotor; Servo ThrustFan; Servo LiftFan; // Transmitter readings. float ReceiverXValue = 0.0; float ReceiverYValue = 0.0; float ReceiverThrotValue = 0.0; void setup() { Serial.begin(9600); XIn.init(0, PIN_X_IN); YIn.init(0, PIN_Y_IN); ThrotIn.init(0, PIN_THROT_IN); LeftRotor.attach(PIN_LEFT_ROTOR); RightRotor.attach(PIN_RIGHT_ROTOR); ThrustFan.attach(PIN_THROT_OUT); LiftFan.attach(PIN_LIFT_OUT); } bool IsPPMInRange(int val) { return ((val <= 2100) && (val >= 900)); } float NormalizePPM(int val) { return ((val - 1000)/1000.0); } float InvertPPM(int val) { return (val - (2*(val - CENTER_PPM))); } // TODO: no magic numbers float LimitPPM(float val) { if (val >= 2000.0) { return 2000.0; } else if (val <= 1000.0) { return 1000.0; } else { return val; } } </pre>	<pre> // TODO: no magic numbers float SafteyPPM(float val) { if (val >= 1750.0) { return 1500.0; } else if (val <= 1250.0) { return 1500.0; } else { return val; } } float AverageFloatArray(float* array, int numVals) { int i; float sum = 0; for (i = 0; i < numVals; i++) { sum = sum + array[i]; } return sum/((float)numVals); } void ReadXSignal() { static float averagedVals[NUM_AVERAGES] = {0}; static int i = 0; int possibleVal = 0; XIn.readSignal(); possibleVal = XIn.getSignal(); if (IsPPMInRange(possibleVal)) { averagedVals[i] = possibleVal; i = (i + 1) % NUM_AVERAGES; ReceiverXValue = AverageFloatArray(averagedVals, NUM_AVERAGES); } } void ReadYSignal() { static float averagedVals[NUM_AVERAGES] = {0}; static int i = 0; int possibleVal = 0; YIn.readSignal(); possibleVal = InvertPPM(YIn.getSignal()); if (IsPPMInRange(possibleVal)) { averagedVals[i] = possibleVal; i = (i + 1) % NUM_AVERAGES; ReceiverYValue = AverageFloatArray(averagedVals, NUM_AVERAGES); } } void ReadThrotSignal() { static float averagedVals[NUM_AVERAGES] = {0}; static int i = 0; </pre>	<pre> int possibleVal = 0; ThrotIn.readSignal(); possibleVal = ThrotIn.getSignal(); if (IsPPMInRange(possibleVal)) { averagedVals[i] = possibleVal; i = (i + 1) % NUM_AVERAGES; ReceiverThrotValue = AverageFloatArray(averagedVals, NUM_AVERAGES); } } void ApplyLeftRotor(float val) { LeftRotor.writeMicroseconds(SafteyPPM(InvertPPM(val))); } void ApplyRightRotor(float val) { RightRotor.writeMicroseconds(SafteyPPM(InvertPPM(val))); } void CalculateServoPositions() { float dx = ReceiverXValue - CENTER_PPM; float dy = ReceiverYValue - CENTER_PPM; float left = LimitPPM(CENTER_PPM + dy + dx); float right = LimitPPM(CENTER_PPM + dy - dx); ApplyLeftRotor(left); ApplyRightRotor(right); } void loop() { // Read all 3 signals from the receiver. ReadXSignal(); ReadYSignal(); ReadThrotSignal(); // Determine the servo positions based on the signal readings. CalculateServoPositions(); // TODO: no magic numbers if (ReceiverThrotValue <= 1000) { LiftFan.writeMicroseconds(1000); ThrustFan.writeMicroseconds(1000); } else { LiftFan.writeMicroseconds(1200); ThrustFan.writeMicroseconds(ReceiverThrotValue); } // Limit the main loop speed for the sake of the servos. //delay(30); } </pre>
---	---	--

Appendix H: Arduino Code Self Stabilizer

<pre> /* Cory D. Hunt, Jose Upegui PID control of hovercraft heading ME445 Final Project*/ #include <Servo.h> //Library for servo control #include<Wire.h> //Library for gyro reading // Identify servos Servo ServoRight; Servo ServoLeft; Servo LiftFan; //Identify variables float ZRateDesired = 0.0; //Desired rate of rotation in deg/sec float errorNew; //Error between desired rotational rate and actual (n) float errorOld; //Error between desired rotational rate and actual (n-1) float dError; //Change in error since last iteration float timeNew; // Time when rate is found (n) float timeOld; // Time from previous iteration (n-1) float dTime; //Change in time since last iteration in microseconds float MidDegree = 94.0; //neutral position of servos float DegreeLeft; //Position that Left servo is commanded to achieve float DegreeRight; //Position that Right servo is commanded to achieve float GyZdegree; //Degree/sec value of gyro data float GyZnew; // Averaged GyZ value float Sum=0.0; //Integral starting point float ResponseProportional; float ResponseDerivative; float ResponseIntegral; float kp=.1; //Proportional gain float kd=.5; //Derivative gain float ki=.005; //Integral gain int NUM_AVERAGES = 8; //Number of gyro readings to average for signal smoothing float averagedVals[8] = {0}; //initialize array size int i = 0; int possibleVal = 0; const int MPU=0x68; // I2C address of the MPU-6050 int 16_t GyZ; //Define gyro reading #define NUMMOTORS 2 typedef struct MotorDef { Servo Motor; int Pin; }; MotorDef Motors[NUMMOTORS]; // Store the settings for both ESCs typedef struct ESCSettingsDef { int Low; int High; int RunThrust; int RunLift; }; ESCSettingsDef ESCSettings; int CurrentSpeed; int Step = 10; #define ESC_HIGH_DEFAULT 200 //Max signal to ESC #define ESC_LOW_DEFAULT 20 //Min signal to ESC #define ESC_RUN_THRUST 50 //Running signal to thrust ESC #define ESC_RUN_LIFT 80 //Running signal to lift fan ESC void setup() { Wire.begin(); Wire.beginTransmission(MPU); Wire.write(0x6B); // PWR_MGMT_1 register Wire.write(0); // set to zero (wakes up the MPU-6050) Wire.endTransmission(true); Serial.begin(9600); Motors[0].Pin = 10; //Thrust motor pin Motors[1].Pin = 11; //Lift motor pin </pre>	<pre> //Attach motors to appropriate pins for(int i = 0; i < NUMMOTORS; i++) { int pin = Motors[i].Pin; Motors[i].Motor.attach(pin); } // Set the ESC settings to the defaults ESCSettings.Low = ESC_LOW_DEFAULT; ESCSettings.High = ESC_HIGH_DEFAULT; ESCSettings.RunThrust = ESC_RUN_THRUST; ESCSettings.RunLift = ESC_RUN_LIFT; /*ESC Arming Procedure: Before the ESC can receive a signal it must be armed. The arming procedure involves sending max signal, waiting for beep confirmation the sending low signal and waiting for beep confirmation*/ Motors[0].Motor.write(ESCSettings.High); Motors[1].Motor.write(ESCSettings.High); delay(6000); Motors[0].Motor.write(ESCSettings.Low); Motors[1].Motor.write(ESCSettings.Low); delay(3000); } void loop() { Wire.beginTransmission(MPU); Wire.write(0x3B); // starting with register 0x3B (ACCEL_XOUT_H) Wire.endTransmission(false); Wire.requestFrom(MPU,14,true); // request a total of 14 registers GyZ=Wire.read()<<8 Wire.read(); // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L) /*The section of code below averages the most recent 8 values of Z axis rotational rate*/ possibleVal = GyZ; averagedVals[i] = possibleVal; i = (i + 1) % NUM_AVERAGES; GyZnew = AverageFloatArray(averagedVals, NUM_AVERAGES); GyZdegree = GyZnew*(250.0/32767.0); //Convert raw gyro reading into deg/sec errorNew = ZRateDesired-GyZdegree; //Find difference between desired and actual rate Sum = Sum+GyZdegree; //Sum for integral response dError = errorNew-errorOld; //Change in error since last iteration dTime = (timeNew-timeOld)*(.0001); //Change in time in seconds ResponseProportional = kp*errorNew; ResponseDerivative = kd*dError/dTime; ResponseIntegral = ki*Sum; DegreeRight = MidDegree - ResponseProportional + ResponseDerivative + ResponseIntegral; DegreeLeft = MidDegree + ResponseProportional - ResponseDerivative - ResponseIntegral; //Save current values for use in next iteration errorOld = errorNew; timeOld = timeNew; //Write signals to servos ServoRight.attach(3); //Right servo to pin 3 ServoLeft.attach(9); //Left servo to pin 9 ServoRight.write((int)LimitDegree(DegreeRight)); ServoLeft.write((int)LimitDegree(DegreeLeft)); //Write signals to thrust and lift ESCs Motors[0].Motor.write(ESCSettings.RunThrust); Motors[1].Motor.write(ESCSettings.RunLift); } //Function for limiting servo travel float LimitDegree(float val) { if (val >= 105.0) { return 105.0; } else if (val <= 85.0) { return 85.0; } else { return val; } } //Function for averaging array of values float AverageFloatArray(float* array, int numVals) { int i; float sum = 0; for (i = 0; i < numVals; i++) { sum = sum + array[i]; } return sum/((float)numVals); } </pre>
--	---

M E 445

FINAL PROJECT REPORT

BoozeStream: A Wireless Breathalyzer Music Player

Department of Mechanical and Nuclear Engineering

The Pennsylvania State University

Submitted by:

Ray McDivitt

Kevin Magee

December 18, 2014

Abstract

This report outlines the creation of a breathalyzer that is wirelessly connected to a music playing device. An alcohol gas sensor is used to gather alcohol concentration data from the air that is then interpreted and transmitted by an attached Arduino Fio through a 2.4GHz wireless tree network. A receiving Arduino Fio is set up to gather and further interpret the data of all of the transmitters and output a running average of that collected data to an attached Arduino Uno. The Uno is equipped with an MP3 shield used to play unique playlists from a micro SD card based on the input from the attached Fio.

Table of Contents

1: Introduction.....	1
2: Hardware.....	2
3: Software	4
4: Device Operation	5
5: Results and Discussion	6
6: Conclusions and Recommendations	7
Appendix A – Pictures of Assembly.....	8
Appendix B – Transmitter Code	10
Appendix C – Receiver Code	13
Appendix D – MP3 Player Code	16
Appendix E – Component Datasheets	20

1: Introduction

Many individuals have had the unfortunate pleasure of getting their blood alcohol level tested at one point or another by means of a policeman's breathalyzer – especially around college campuses. While there is a negative connotation associated with these devices, the way they work is quite interesting and complex. The inspiration behind this project comes from the amazing functionality of a breathalyzer and the possibilities to enhance its quality and entertainment value.

The goal of this project was to create a breathalyzer that is wirelessly connected to a music device through a tree network and will play unique music depending on the level of intoxication. Normal breathalyzers simply display BAC data as numbers. This breathalyzer displays BAC data by playing a song.

The breathalyzer uses an MQ-3 alcohol sensor connected to an Arduino Fio to gather BAC data from a person's breath. The BAC data is then wirelessly transmitted through an nRF24L01 transceiver, which uses the same 2.4GHz network that most Wi-Fi routers use, to a receiving Arduino Fio. The receiving Arduino Fio gathers the data from all of the transmitters and maintains a running average of the BAC level in the room. It then outputs the BAC level to an Arduino Uno with an Adafruit MP3 shield attached to it. This shield allows the incoming BAC data to be expressed as unique playlists for each of the five BAC levels. Nine songs were chosen and set up as MP3 files to play depending on the input data received by Uno. For example, "My Life Be Like" by Grits will play if the receiving Fio outputs a one, which is between a zero and roughly 0.05 BAC. If the BAC average remains the same throughout the time that that song is playing, the the "Team America: World Police" theme song will start playing. Each range of BAC data is matched with what best captures the probable "mood" of that person. For example, the first song in the playlist for the second highest range is "Drunken Sailor" by Irish Rovers.

We chose this project for three main reasons: the challenge, the marketability, and the real world applications. The coding required and overall effort to get wireless transmission operating as well as getting everything working together all at once was very difficult and offered a great opportunity to increase our knowledge base and expand our own personal horizons. The marketability of a product like this is incredible. College students would absolutely love this, and one can imagine a much more refined and cost-friendly product similar to this that was actually on the market would sell extremely well. This project also has significant real-world applications. Collecting BAC data and outputting it reliably is useful in the real world if you want to take a stronger stand against drunk driving. The most useful real world application of this project is undoubtedly the use of a cheap, reliable wireless tree network that can be utilized in a large variety of projects. These networks can be used to connect thousands of devices together to maintain constant communication amongst all of them.

2: Hardware

<i>Transmitter</i>			<i>Receiver</i>		
<u>Item</u>	<u>Quantity</u>	<u>Cost</u>	<u>Item</u>	<u>Quantity</u>	<u>Cost</u>
Arduino Fio	1	\$24.95	Arduino Fio	1	\$24.95
nRF24L01+ Transceiver	1	\$3.75	nRF24L01+ Transceiver	1	\$3.75
7805CT MOSFET	1	\$0.48	47K Ω Resistor	3	\$0.05
MQ-3 Alcohol Sensor	1	\$4.95	8GB MicroSD Card	1	\$4.99
10K Ω Resistor	1	\$0.05	Adafruit "Music Maker" MP3 Shield	1	\$24.95
		\$34.18	Arduino Uno	1	\$14.41
					\$73.20
<i>Other Hardware</i>					
FTDI Breakout Board	1	\$14.95			
Batteries	(varies)	(varies)			

Table 1: Bill of Materials

There are two primary components to this project, the transmitters and the receiver. The transmitter is used to gather and transmit BAC data. An Arduino Fio is used as the controller for the system. There is a regulated 5V input that is used to power the Fio as well as the MQ-3 alcohol gas sensor. The sensor requires that the heater core be powered and grounded through the 5V source to get it up to temperature to provide necessary work conditions. A measuring electrode provides a variable voltage output that can be read through an analog input on the Fio and calibrated for different BAC levels. A 10K Ω pull down resistor is used to measure a reasonable range of alcohol concentration. The nRF24L01+ transceiver is wired to the Fio and set up as a leaf within the tree network that is transmitting data directly to the base. Figure 1 shows a breakdown of how tree topology looks.

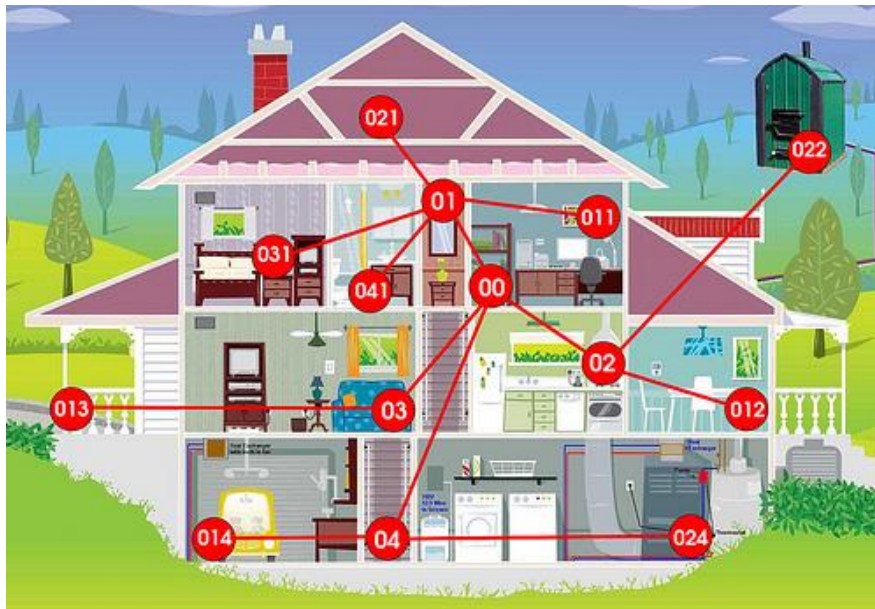


Figure 1: Tree Topology

In order to send data to the proper node within the tree network, each node must be assigned a 15-bit address. This address exactly describes the position of the node within the tree. The address is an octal number where each digit in the address represents a position in the tree further from the base. For this type of network:

- Node 00 is the base node.
- Nodes 01-05 are nodes whose parent is the base.
- Node 021 is the second child of node 01.
- Node 0321 is the third child of node 021, and so on.
- The largest node address is 05555, so 3,125 nodes are allowed on a single channel.

The receiver also uses an Arduino Fio with an nRF24L01+ transceiver, but it is set up as the base in the network. The RF24 is not connected directly to the Arduino Uno and MP3 shield because the shield uses pins that are required by the RF24 to operate. In order to transmit the BAC averages from the Fio to the Uno, a binary pin setup is utilized. Three pins on the Fio are wired to the Uno/MP3 shield bundle through a set of pull down resistors shown in Figure 2.

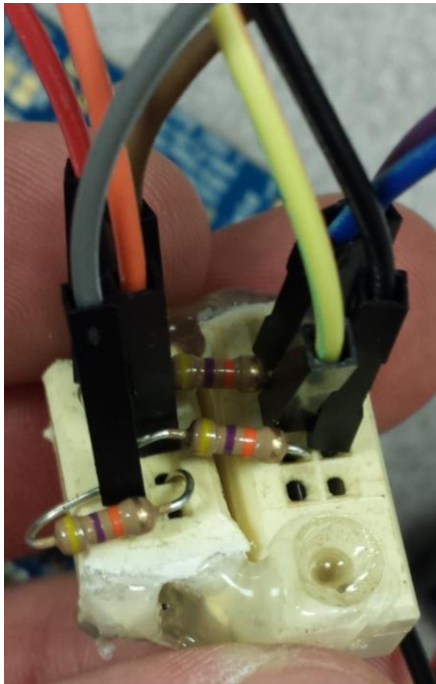


Figure 2: Pull Down Resistor Set

The Uno/MP3 shield bundle interprets the binary inputs through the pins that correspond to the Fio output. There is an SD card slot on the MP3 shield with a card that is loaded with songs used for the various playlists that are called out by the Uno. The MP3 shield has a 3.5mm headphone jack that is hooked up to a set of speakers for playing the music.

3: Software

A detailed version of our code is available Appendix B, C, and D. Figure 3 is a rough flow diagram of the software.

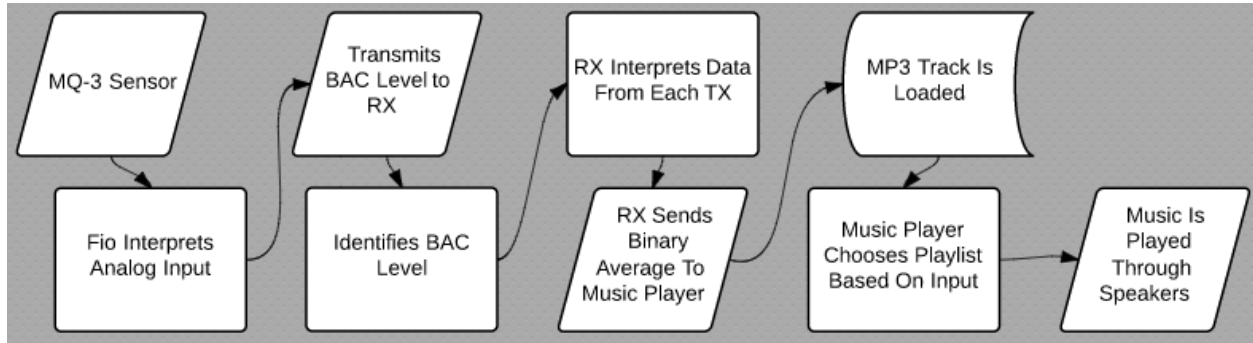


Figure 3: Software Flow Diagram

4: Device Operation

Proper operation of the device is as follows:

1. Load the SD card with tracks with the title “track001” through “track010” with a MP3 file extension. The file name within the file properties should be “track0XX.mp3”
2. Insert the SD card into the MP3 shield.
3. Plug a set of speakers or headphones into the MP3 shield.
4. Power up the TX lapel pins and the Uno/MP3 shield bundle. They will now be gathering data and music will start playing.
5. As data is gathered, different playlists will be loaded and start playing. Nothing else needs to be done as the system will continue to gather data and change as time goes on.

5: Results and Discussion

Throughout this project, multiple issues were encountered. The primary issue was getting the RF24L01+ tree network up and running. There are a variety of libraries available for use with the transceivers, each at different stages of development and with slightly different methods of operation. After an extensive search and test process, the library by tmrh20 available at <https://github.com/tmrh20/RF24> was chosen for its improved functionality and fluid operation. This library is able to theoretically handle the networking of thousands of nodes efficiently and accurately.

Another issue that is inherent to the project without extensive real world testing is the use of the MQ-3 alcohol gas sensor. It is meant to be used with breathalyzer devices where the user has to intentionally breathe into the sensor for an extended period of time. This allows the sensor to gather sufficient, accurate data. In this project, the sensor passively gathers data throughout conversations. This means that the data isn't the most accurate, but it is still sufficient. The system needs to be properly calibrated for a variety of environments and venues. For instance, an apartment will transmit the alcohol on people's breaths differently than a pool party. Gathering this data would result in a more accurate, useful system as a whole.

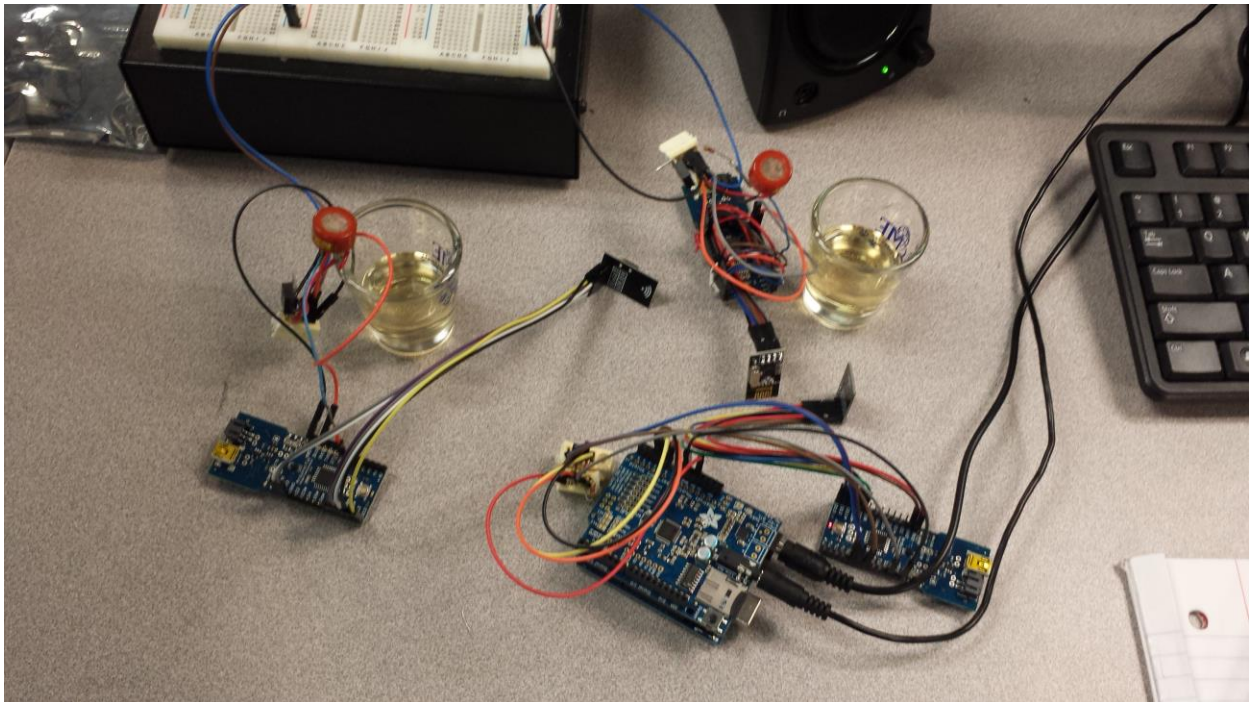
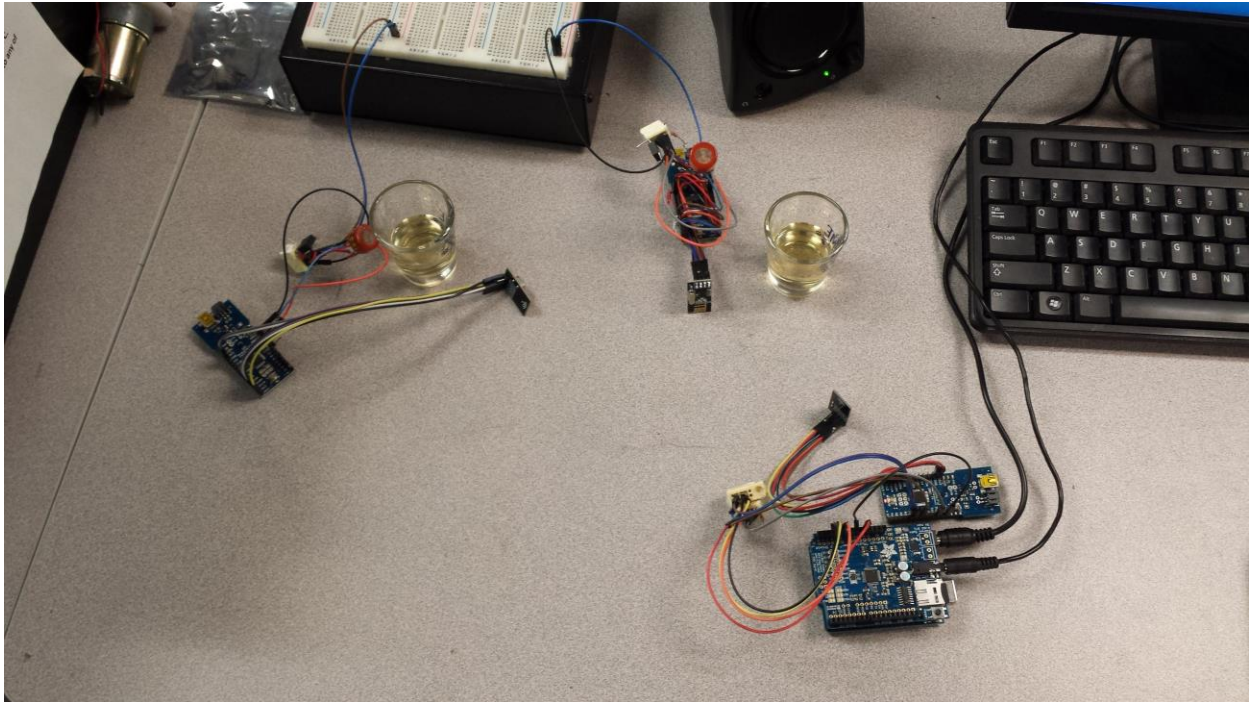
Despite complications, the system operated exactly as expected. When holding alcohol near the sensors, it was able to gather alcohol concentration data, transmit it through the network, output it to the MP3 player, and accurately play music for the different levels received.

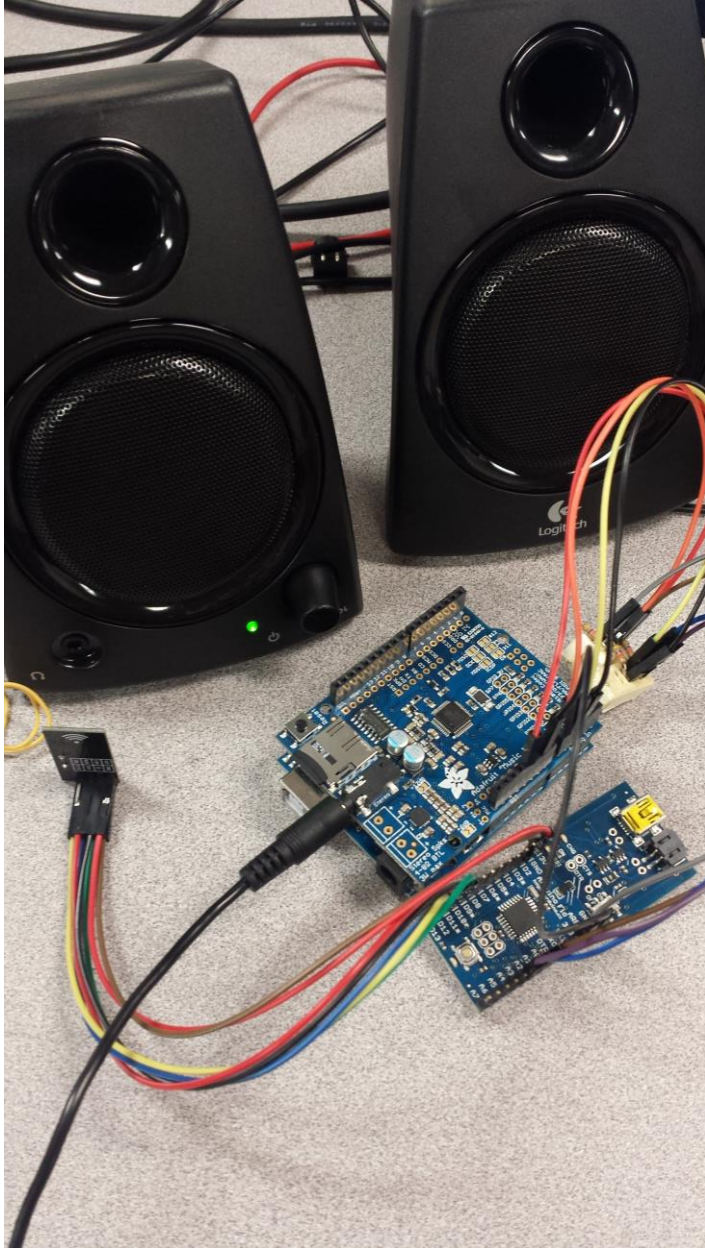
6: Conclusions and Recommendations

The project came together very nicely in the end. Everything worked as expected and there is little that would be changed to create a final product. In reality, with the hardware we had available, the only thing left to do would be to put each of the parts of the project into proper packaging. If we were to go further with it, an optimized setup could be developed that had just the right amount of components, resulting in a compact solution for the entire project. If anyone were to do a project that requires networking, utilizing the nRF24L01+ transceivers is highly recommended, especially when used in conjunction with the library by tmrh20. They are a feasible replacement to the XBee transceivers that are commonly used with Arduino projects.

Given how this project worked exactly as expected, it was a huge success. Not only was it a success from the point of view of the project working, but it was also a success from a personal point of view. We were able to work through all of the issues encountered and create a properly functioning project. Everything worked together in unison without problems. As mechanical engineers, successfully completing a project with a major focus on programming made us feel extremely accomplished. It may not have been the flashiest project, but it works, and it works well. As Steve Jobs said, “beauty lies in simplicity.” This project is an embodiment of that quote and exceeded our expectations.

Appendix A – Pictures of Assembly





Appendix B – Transmitter Code

```
/*
This program is used for each Lapel Pin Alcohol Sensor. It will read the MQ-
3 sensor
data and send that data to the Arduino receiver. Data is polled every 30 seconds.
(MQ-3 Calibrating: http://nootropicdesign.com/projectlab/2010/09/17/arduino-breathalyzer/)

Board: Arduino Fio
(requires FT231X Breakout https://www.sparkfun.com/products/11736, which we have)

Wiring:
MQ-3 (http://wiring.org.co/learning/basics/airqualitymq135.html):
Analog pin A0 to MQ-3 input
MQ-3 GND-B to 10k ohm resistor or 100k ohm potentiometer to ground
MQ-3 GND-H to ground
MQ-3 A side to external +5V (7805 voltage regulated)
-OR-
H1 to 5V DC
A1 to 5V DC
GND to GND
B1 with 10K ohm to GND and wire to A0 on Arduino (B1 -> wire to arduino -
> 10K to GND)

Wireless Module:
VCC to 3.3V Arduino
GND to GND Arduino
Pin 9 to CE
Pin 10 to CSN
Pin 11 to MOSI
Pin 12 to MISO
Pin 13 to SCK
Libraries at https://github.com/tmrh20/RF24

Ray McDivitt

***** modify "id" before uploading to each TX *****
***** 1 - 5 for first children *****
*/

//Often changed variables:
//identifier
unsigned int id = 1;
//address of this node
uint16_t this_node = 01;
//sets threshold for BAC levels
int bacMin = 50; //100 for node 2, 50 for node 1
//used for timer (gathered for totalTime, idle for delayTime)
const int totalTime = 15000;
const int delayTime = 15000;
//for BAC mapping
const int maxBAC = 6;
```

```

//For Wireless:
//RF24 libraries
#include <nRF24L01.h>
#include <RF24.h>
#include <RF24_config.h>
#include <RF24Network.h>
#include <RF24Network_config.h>
#include <Sync.h>
#include <SPI.h>
//Sets up RF24 on SPI bus pins 9 & 10
RF24 radio(9,10);
//Network uses that radio
RF24Network network(radio);
//Where to send node
uint16_t to_node = 00;
//structure of data sent
struct packet_t {
    //TX identifier
    unsigned int ident;
    //BAC level
    unsigned int level;
};

//For MQ-3

const int analogPin = A0;    // the pin that the MQ-3 sensor
//ints used for calculating running average BAC
unsigned int avgBAC,bacLVL;
//ints used for readings
unsigned int firstReading,currentReading;
//longs used for timing
unsigned long beginTime,currentTime,endTime;

void setup() {
    Serial.begin(57600);
    pinMode(analogPin,INPUT);

    SPI.begin();
    radio.begin();
    network.begin(/*channel*/ 90,/*node address*/ this_node);
}

void loop() {
    //timer setup
    beginTime = millis();
    // Serial.print("Begin time: ");
    // Serial.println(beginTime);
    currentTime = beginTime;
    endTime = beginTime + totalTime;
    // Serial.print("End time: ");
    // Serial.println(endTime);

    //data gathering for totalTime
    //gets an initial reading
    firstReading = analogRead(analogPin);

```

```

    //initializes average
    avgBAC = firstReading;
    // Serial.print("First reading: ");
    // Serial.println(avgBAC);
    while(currentTime < endTime) {
        //gets current reading of BAC
        currentReading = analogRead(analogPin);
        //uses the current reading for average BAC level calculation
        bacLVL = currentReading;
        //calculates new BAC lvl average
        avgBAC = (avgBAC+bacLVL)/2;
    // Serial.println(avgBAC);
        //50ms delay for normalized data
        delay(50);
        currentTime=millis();
    }
    // Serial.print("Raw average BAC: ");
    // Serial.println(avgBAC);
    //averaged BACs mapped to the number of different BAC levels
    avgBAC = map(avgBAC,bacMin,1023,1,maxBAC);
    // Serial.print("Mapped average: ");
    // Serial.println(avgBAC);

    //Send BAC levels
    //continuously check network
    network.update();
    //This is the code to send the sensor data to the RX
    //referencing http://maniacbug.wordpress.com/2012/03/30/rf24network/
    // Serial.println("Sending...");
    //structured packet of data to send
    packet_t packet = {id,avgBAC};
    // Serial.print("ID: ");
    // Serial.println(packet.ident);
    // Serial.print("BAC: ");
    // Serial.println(packet.level);
    RF24NetworkHeader header(/*to node*/ to_node);
    boolean ok = false;
    //continuously tries to send BAC level to receiver until it is successful
    while(ok == false){
        ok = network.write(header,&packet,sizeof(packet));
    // Serial.println("Failed.");
    }
    // Serial.println("Successful.");

    //Delay before gathering next dataset
    delay(delayTime);
}

```

Appendix C – Receiver Code

```
/*
This program is used for the main receiver.

Polling data (sending a "stop" command from the RX to each TX)

Board: Arduino Fio

Wiring:
RF24 (http://arduino-info.wikispaces.com/Nrf24L01-2.4GHz-HowTo)
Pin 9 to CE
Pin 10 to CSN
Pin 11 to MOSI
Pin 12 to MISO
Pin 13 to SCK
Libraries at https://github.com/tmrh20/RF24

Ray McDivitt
*/
//For wireless:
#include <nRF24L01.h>
#include <RF24.h>
#include <RF24_config.h>
#include <RF24Network.h>
#include <RF24Network_config.h>
#include <Sync.h>
#include <SPI.h>

//declares RX as base node
unsigned int id = 00;
//Sets up nRF24L01 radio on SPI bus pins 9 & 10
RF24 radio(9,10);
//Network uses that radio
RF24Network network(radio);
//Address of our node
uint16_t this_node = 00;
//structure of data received
struct packet_t {
    //TX identifier
    unsigned int ident;
    //BAC level
    unsigned int level;
};

//intializes averages and sets it in the middle
int avg;
int avg1 = 1;
int avg2 = 1;

//For playing audio:
//used to do a binary output of average BAC (avg)
const int audCount = 3;
const int audPins[] = {14,15,16};
//int low = 300;
//int high = 1000;
```

```

void setup() {
  // initialize serial communication:
  Serial.begin(57600);
  //enable audio control pins
  for(int thisPin = 0; thisPin < audCount; thisPin++) {
    pinMode(audPins[thisPin], OUTPUT);
    digitalWrite(audPins[thisPin], LOW);
  }

  SPI.begin();
  radio.begin();
  network.begin(/*channel*/ 90, /*node address*/ this_node);
}

void loop() {
  //check the network regularly
  network.update();

  while(network.available())
  {
    RF24NetworkHeader header;
    packet_t packet;
    network.read(header, &packet, sizeof(packet));
    //prints
    Serial.print("Received BAC of level ");
    Serial.print(packet.level);
    Serial.print(" from transmitter #");
    Serial.println(packet.ident);

    //performs operation with gathered data
    //averages BAC levels of all sensors
    if(packet.ident==1){
      avg1 = packet.level;
      Serial.print("first packet: ");
      Serial.println(avg1);
    }
    else if(packet.ident==2){
      avg2 = packet.level;
      Serial.print("second packet: ");
      Serial.println(avg2);
    }
    avg = (avg1+avg2)/2;
    Serial.print("Average: ");
    Serial.println(avg);
    //operations that the RX does
    if(avg==1){
      //Binary of 1
      digitalWrite(audPins[0], LOW);
      digitalWrite(audPins[1], LOW);
      digitalWrite(audPins[2], HIGH);
    }
    else if(avg==2){
      //binary of 2
      digitalWrite(audPins[0], LOW);
      digitalWrite(audPins[1], HIGH);
    }
  }
}

```

```

        digitalWrite(audPins[2],LOW);
    }
    else if (avg==3) {
        //binary of 4
        digitalWrite(audPins[0],LOW);
        digitalWrite(audPins[1],HIGH);
        digitalWrite(audPins[2],HIGH);
    }
    else if (avg==4) {
        //binary of 4
        digitalWrite(audPins[0],HIGH);
        digitalWrite(audPins[1],LOW);
        digitalWrite(audPins[2],LOW);
    }
    else if (avg==5) {
        //binary of 5
        digitalWrite(audPins[0],HIGH);
        digitalWrite(audPins[1],LOW);
        digitalWrite(audPins[2],HIGH);
    }
    else {
        //binary of 6, shouldn't ever happen
        digitalWrite(audPins[0],HIGH);
        digitalWrite(audPins[1],HIGH);
        digitalWrite(audPins[2],LOW);
    }
}
}

```

Appendix D – MP3 Player Code

```
//For Adafruit MP3 shield:
// include SPI, MP3 and SD libraries
#include <SPI.h>
#include <Adafruit_VS1053.h>
#include <SD.h>

// These are the pins used for the music maker shield
#define SHIELD_RESET -1      // VS1053 reset pin (unused!)
#define SHIELD_CS    7      // VS1053 chip select pin (output)
#define SHIELD_DCS    6      // VS1053 Data/command select pin (output)

// These are common pins between breakout and shield
#define CARDCS 4           // Card chip select pin
// DREQ should be an Int pin, see http://arduino.cc/en/Reference/attachInterrupt
#define DREQ 3             // VS1053 Data request (an Interrupt pin)
// create shield-example object!
Adafruit_VS1053_FilePlayer musicPlayer =
  Adafruit_VS1053_FilePlayer(SHIELD_RESET, SHIELD_CS, SHIELD_DCS, DREQ, CARDCS);

//For handling master receiver output:
const int audCount = 3;
const int audPins[] = {14, 15, 16};
//used for mapping binary output to a value
int audValue;
//used for averaging BAC while playing tracks
int bacAvg=1;

void setup() {
  Serial.begin(9600);
  Serial.println("Adafruit VS1053 Library Test");

  // initialise the music player
  if (!musicPlayer.begin()) { // initialise the music player
    Serial.println(F("Couldn't find VS1053, do you have the right pins
defined?"));
    while (1);
  }
  Serial.println(F("VS1053 found"));

  // musicPlayer.sineTest(0x44, 500); // Make a tone to indicate VS1053 is
  // working

  if (!SD.begin(CARDCS)) {
    Serial.println(F("SD failed, or not present"));
    while (1); // don't do anything more
  }
  Serial.println("SD OK!");

  // list files
  printDirectory(SD.open("/"), 0);

  // Set volume for left, right channels. lower numbers == louder volume!
```

```

musicPlayer.setVolume(0,0);

// This option uses a pin interrupt. No timers required! But DREQ
// must be on an interrupt pin. For Uno/Duemilanove/Diecimilla
// that's Digital #2 or #3
// See http://arduino.cc/en/Reference/attachInterrupt for other pins
// *** This method is preferred
if (! musicPlayer.useInterrupt(VS1053_FILEPLAYER_PIN_INT))
    Serial.println(F("DREQ pin is not an interrupt pin"));

//enable audio control pins
for(int thisPin = 0; thisPin < audCount; thisPin++) {
    pinMode(audPins[thisPin], INPUT);
    digitalWrite(audPins[thisPin], LOW);
}
//starts playing intro track
musicPlayer.startPlayingFile("track010.mp3"); //Maroon 5 - Daylight
delay(2000);
}

void loop() {
    // Alternately, we can just play an entire file at once
    // This doesn't happen in the background, instead, the entire
    // file is played and the program will continue when it's done!
    //musicPlayer.playFullFile("track002.mp3");

    // Start playing a file, then we can do stuff while waiting for it to
    finish
    // if (! musicPlayer.startPlayingFile("track009.mp3")) {
    //     Serial.println("Could not open file track00X.mp3");
    //     while (1);
    // }
    // Serial.println(F("Started playing"));

    // while(musicPlayer.playingMusic){
    //     audValue = BinConvert();
    // }
    //gets the current BAC average
    audValue = BinConvert();
    Serial.print("Input value: ");
    Serial.println(audValue);
    //first input for bacAvg
    bacAvg = audValue;

    //starts playing corresponding file for BAC average
    if(audValue ==0){
        delay(1000);
    }
    if(audValue == 1) {
        musicPlayer.startPlayingFile("track001.mp3"); //The Grits - My Life Be
        Like
        //lets the full track play through while gathering a running average of
        the
        // current BAC average
        while(musicPlayer.playingMusic){
            audValue = BinConvert();

```



```

//      Serial.print("Reading: ");
//      Serial.println(audValue);
    bacAvg = (bacAvg + audValue)/2;
    Serial.print("BAC average: ");
    Serial.println(bacAvg);
    delay(500);
}
//keeps playing the same BAC level if the running average remained the
same
    if(bacAvg==1)
        musicPlayer.startPlayingFile("track002.mp3"); //America, Fuck Yeah
    }

    else if(audValue == 2) {
        musicPlayer.startPlayingFile("track003.mp3"); //AC/DC - Back In Black
        while(musicPlayer.playingMusic){
            audValue = BinConvert();
            bacAvg = (bacAvg + audValue)/2;
            delay(500);
        }
        if(bacAvg==2)
            musicPlayer.startPlayingFile("track004.mp3"); //Blink 182 - Feeling
This
    }

    else if(audValue == 3) {
        musicPlayer.startPlayingFile("track005.mp3"); //Tiesto - Wasted
        while(musicPlayer.playingMusic){
            audValue = BinConvert();
            bacAvg = (bacAvg + audValue)/2;
            delay(500);
        }
        if(bacAvg==3)
            musicPlayer.startPlayingFile("track006.mp3"); //Flosstradamus - Too
Turnt Up
    }

    else if(audValue == 4) {
        musicPlayer.startPlayingFile("track007.mp3"); //Dropkick Murphys -
Shipping Up To Boston
        while(musicPlayer.playingMusic){
            audValue = BinConvert();
            bacAvg = (bacAvg + audValue)/2;
            delay(500);
        }
        if(bacAvg==4)
            musicPlayer.startPlayingFile("track008.mp3"); //Irish Rovers - Drunken
Sailor
    }

    else if(audValue == 5) {
        musicPlayer.startPlayingFile("track009.mp3"); //O Fortuna
    }
    else{
        musicPlayer.startPlayingFile("track010.mp3"); //Maroon 5 - Daylight
    }

    //waits for last track in playlist to finish playing

```

```

    while(musicPlayer.playingMusic)
        delay(500);
}

/// File listing helper
void printDirectory(File dir, int numTabs) {
    while(true) {

        File entry = dir.openNextFile();
        if (! entry) {
            // no more files
            //Serial.println("**nomorefiles**");
            break;
        }
        for (uint8_t i=0; i<numTabs; i++) {
            Serial.print('\t');
        }
        Serial.print(entry.name());
        if (entry.isDirectory()) {
            Serial.println("/");
            printDirectory(entry, numTabs+1);
        } else {
            // files have sizes, directories do not
            Serial.print("\t\t");
            Serial.println(entry.size(), DEC);
        }
        entry.close();
    }
}

int BinConvert()
{
    int sum = 0;
    if(digitalRead(audPins[0])==HIGH)
        sum+=4;
    if(digitalRead(audPins[1])==HIGH)
        sum+=2;
    if(digitalRead(audPins[2])==HIGH)
        sum+=1;
    return(sum);
}

```

Appendix E – Component Datasheets

Relevant component datasheets are attached in the pages following this. The MP3 shield has an extensive datasheet available at <http://www.adafruit.com/datasheets/vs1053.pdf>

nRF24L01+

Single Chip 2.4GHz Transceiver

Product Specification v1.0

Key Features

- Worldwide 2.4GHz ISM band operation
- 250kbps, 1Mbps and 2Mbps on air data rates
- Ultra low power operation
- 11.3mA TX at 0dBm output power
- 13.5mA RX at 2Mbps air data rate
- 900nA in power down
- 26µA in standby-I
- On chip voltage regulator
- 1.9 to 3.6V supply range
- Enhanced ShockBurst™
- Automatic packet handling
- Auto packet transaction handling
- 6 data pipe MultiCeiver™
- Drop-in compatibility with nRF24L01
- On-air compatible in 250kbps and 1Mbps with nRF2401A, nRF2402, nRF24E1 and nRF24E2
- Low cost BOM
- ±60ppm 16MHz crystal
- 5V tolerant inputs
- Compact 20-pin 4x4mm QFN package

Applications

- Wireless PC Peripherals
- Mouse, keyboards and remotes
- 3-in-1 desktop bundles
- Advanced Media center remote controls
- VoIP headsets
- Game controllers
- Sports watches and sensors
- RF remote controls for consumer electronics
- Home and commercial automation
- Ultra low power sensor networks
- Active RFID
- Asset tracking systems
- Toys

Liability disclaimer

Nordic Semiconductor ASA reserves the right to make changes without further notice to the product to improve reliability, function or design. Nordic Semiconductor ASA does not assume any liability arising out of the application or use of any product or circuits described herein.

All application information is advisory and does not form part of the specification.

Limiting values

Stress above one or more of the limiting values may cause permanent damage to the device. These are stress ratings only and operation of the device at these or at any other conditions above those given in the specifications are not implied. Exposure to limiting values for extended periods may affect device reliability.

Life support applications

These products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Nordic Semiconductor ASA customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Nordic Semiconductor ASA for any damages resulting from such improper use or sale.

Data sheet status	
Objective product specification	This product specification contains target specifications for product development.
Preliminary product specification	This product specification contains preliminary data; supplementary data may be published from Nordic Semiconductor ASA later.
Product specification	This product specification contains final product specifications. Nordic Semiconductor ASA reserves the right to make changes at any time without notice in order to improve design and supply the best possible product.

Contact details

Visit www.nordicsemi.no for Nordic Semiconductor sales offices and distributors worldwide

Main office:

Otto Nielsens vei 12
7004 Trondheim
Phone: +47 72 89 89 00
Fax: +47 72 89 89 89
www.nordicsemi.no



Writing Conventions

This product specification follows a set of typographic rules that makes the document consistent and easy to read. The following writing conventions are used:

- Commands, bit state conditions, and register names are written in `Courier`.
- Pin names and pin signal conditions are written in **`Courier bold`**.
- Cross references are [underlined and highlighted in blue](#).

Revision History

Date	Version	Description
September 2008	1.0	

Attention!

Observe precaution for handling
Electrostatic Sensitive Device.

HBM (Human Body Model) $\geq 1\text{Kv}$
MM (Machine Model) $\geq 200\text{V}$



Contents

1	Introduction	7
1.1	Features	8
1.2	Block diagram	9
2	Pin Information	10
2.1	Pin assignment.....	10
2.2	Pin functions.....	11
3	Absolute maximum ratings	12
4	Operating conditions	13
5	Electrical specifications	14
5.1	Power consumption.....	14
5.2	General RF conditions	15
5.3	Transmitter operation	15
5.4	Receiver operation	16
5.5	Crystal specifications	19
5.6	DC characteristics	20
5.7	Power on reset	20
6	Radio Control	21
6.1	Operational Modes	21
6.1.1	State diagram	21
6.1.2	Power Down Mode	22
6.1.3	Standby Modes.....	22
6.1.4	RX mode.....	23
6.1.5	TX mode	23
6.1.6	Operational modes configuration.....	24
6.1.7	Timing Information	24
6.2	Air data rate.....	25
6.3	RF channel frequency	25
6.4	Received Power Detector measurements.....	25
6.5	PA control.....	26
6.6	RX/TX control	26
7	Enhanced ShockBurst™	27
7.1	Features	27
7.2	Enhanced ShockBurst™ overview.....	27
7.3	Enhanced Shockburst™ packet format.....	28
7.3.1	Preamble	28
7.3.2	Address	28
7.3.3	Packet control field	28
7.3.4	Payload.....	29
7.3.5	CRC (Cyclic Redundancy Check)	30
7.3.6	Automatic packet assembly	31
7.3.7	Automatic packet disassembly	32
7.4	Automatic packet transaction handling	33
7.4.1	Auto acknowledgement	33
7.4.2	Auto Retransmission (ART).....	33

7.5	Enhanced ShockBurst flowcharts	35
7.5.1	PTX operation.....	35
7.5.2	PRX operation	37
7.6	MultiCeiver™	39
7.7	Enhanced ShockBurst™ timing	42
7.8	Enhanced ShockBurst™ transaction diagram	45
7.8.1	Single transaction with ACK packet and interrupts.....	45
7.8.2	Single transaction with a lost packet	46
7.8.3	Single transaction with a lost ACK packet	46
7.8.4	Single transaction with ACK payload packet	47
7.8.5	Single transaction with ACK payload packet and lost packet.....	47
7.8.6	Two transactions with ACK payload packet and the first ACK packet lost	48
7.8.7	Two transactions where max retransmissions is reached	48
7.9	Compatibility with ShockBurst™	49
7.9.1	ShockBurst™ packet format	49
8	Data and Control Interface	50
8.1	Features	50
8.2	Functional description	50
8.3	SPI operation	50
8.3.1	SPI commands	50
8.3.2	SPI timing	52
8.4	Data FIFO	55
8.5	Interrupt.....	56
9	Register Map.....	57
9.1	Register map table	57
10	Peripheral RF Information	64
10.1	Antenna output.....	64
10.2	Crystal oscillator.....	64
10.3	nRF24L01+ crystal sharing with an MCU.....	64
10.3.1	Crystal parameters	64
10.3.2	Input crystal amplitude and current consumption	64
10.4	PCB layout and decoupling guidelines.....	65
11	Application example	66
11.1	PCB layout examples	67
12	Mechanical specifications.....	71
13	Ordering information	73
13.1	Package marking	73
13.2	Abbreviations	73
13.3	Product options	73
13.3.1	RF silicon	73
13.3.2	Development tools	73
14	Glossary of Terms.....	74
	Appendix A - Enhanced ShockBurst™ - Configuration and communication example	75
	Enhanced ShockBurst™ transmitting payload.....	75

Enhanced ShockBurst™ receive payload	76
Appendix B - Configuration for compatibility with nRF24XX.....	77
Appendix C - Constant carrier wave output for testing.....	78
Configuration	78

1 Introduction

The nRF24L01+ is a single chip 2.4GHz transceiver with an embedded baseband protocol engine (Enhanced ShockBurst™), suitable for ultra low power wireless applications. The nRF24L01+ is designed for operation in the world wide ISM frequency band at 2.400 - 2.4835GHz.

To design a radio system with the nRF24L01+, you simply need an MCU (microcontroller) and a few external passive components.

You can operate and configure the nRF24L01+ through a Serial Peripheral Interface (SPI). The register map, which is accessible through the SPI, contains all configuration registers in the nRF24L01+ and is accessible in all operation modes of the chip.

The embedded baseband protocol engine (Enhanced ShockBurst™) is based on packet communication and supports various modes from manual operation to advanced autonomous protocol operation. Internal FIFOs ensure a smooth data flow between the radio front end and the system's MCU. Enhanced ShockBurst™ reduces system cost by handling all the high speed link layer operations.

The radio front end uses GFSK modulation. It has user configurable parameters like frequency channel, output power and air data rate. nRF24L01+ supports an air data rate of 250 kbps, 1 Mbps and 2Mbps. The high air data rate combined with two power saving modes make the nRF24L01+ very suitable for ultra low power designs.

nRF24L01+ is drop-in compatible with nRF24L01 and on-air compatible with nRF2401A, nRF2402, nRF24E1 and nRF24E2. Intermodulation and wideband blocking values in nRF24L01+ are much improved in comparison to the nRF24L01 and the addition of internal filtering to nRF24L01+ has improved the margins for meeting RF regulatory standards.

Internal voltage regulators ensure a high Power Supply Rejection Ratio (PSRR) and a wide power supply range.

1.1 Features

Features of the nRF24L01+ include:

- Radio
 - Worldwide 2.4GHz ISM band operation
 - 126 RF channels
 - Common RX and TX interface
 - GFSK modulation
 - 250kbps, 1 and 2Mbps air data rate
 - 1MHz non-overlapping channel spacing at 1Mbps
 - 2MHz non-overlapping channel spacing at 2Mbps
- Transmitter
 - Programmable output power: 0, -6, -12 or -18dBm
 - 11.3mA at 0dBm output power
- Receiver
 - Fast AGC for improved dynamic range
 - Integrated channel filters
 - 13.5mA at 2Mbps
 - -82dBm sensitivity at 2Mbps
 - -85dBm sensitivity at 1Mbps
 - -94dBm sensitivity at 250kbps
- RF Synthesizer
 - Fully integrated synthesizer
 - No external loop filter, VCO varactor diode or resonator
 - Accepts low cost ± 60 ppm 16MHz crystal
- Enhanced ShockBurst™
 - 1 to 32 bytes dynamic payload length
 - Automatic packet handling
 - Auto packet transaction handling
 - 6 data pipe MultiCeiver™ for 1:6 star networks
- Power Management
 - Integrated voltage regulator
 - 1.9 to 3.6V supply range
 - Idle modes with fast start-up times for advanced power management
 - 26 μ A Standby-I mode, 900nA power down mode
 - Max 1.5ms start-up from power down mode
 - Max 130 μ s start-up from standby-I mode
- Host Interface
 - 4-pin hardware SPI
 - Max 10Mbps
 - 3 separate 32 bytes TX and RX FIFOs
 - 5V tolerant inputs
- Compact 20-pin 4x4mm QFN package

1.2 Block diagram

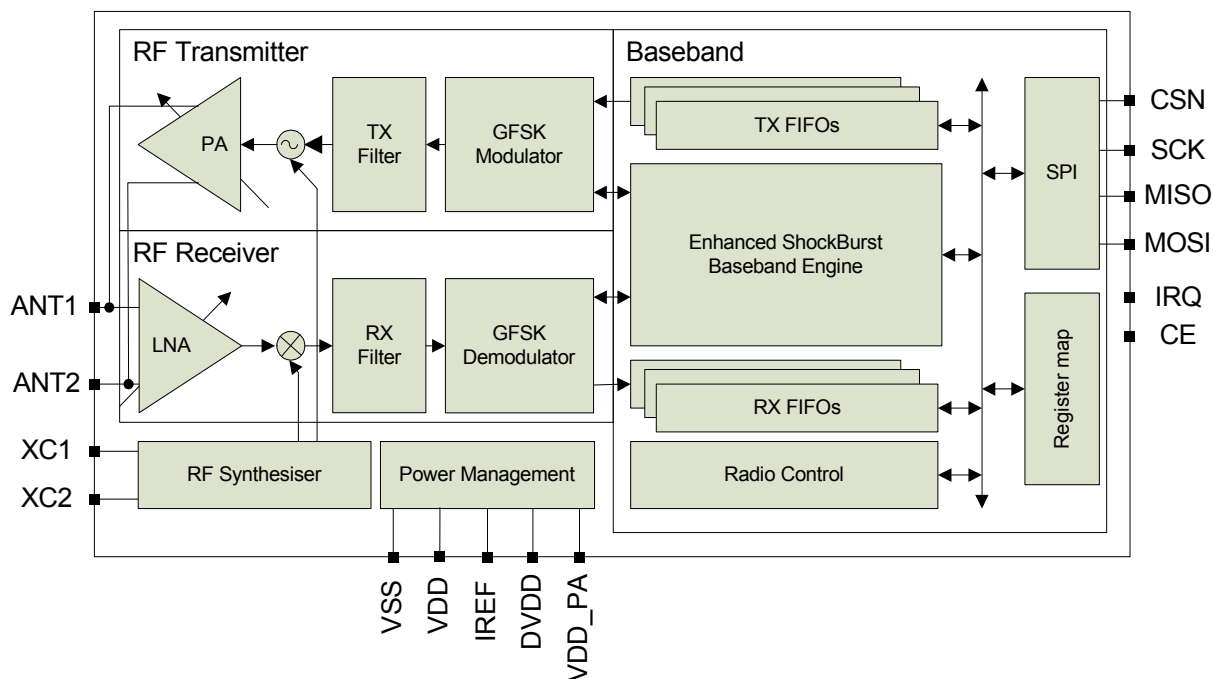


Figure 1. nRF24L01+ block diagram

2 Pin Information

2.1 Pin assignment

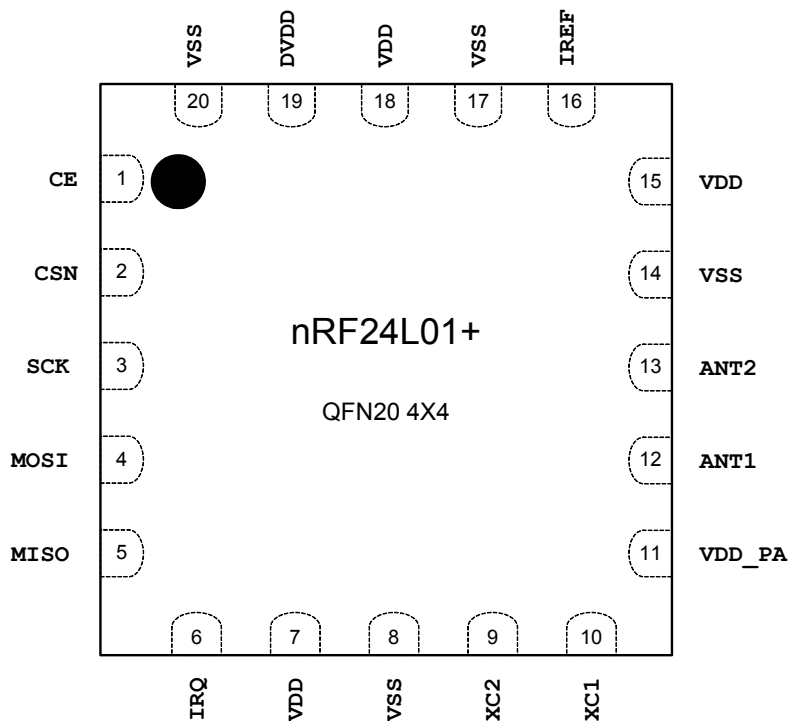


Figure 2. nRF24L01+ pin assignment (top view) for the QFN20 4x4 package

2.2 Pin functions

Pin	Name	Pin function	Description
1	CE	Digital Input	Chip Enable Activates RX or TX mode
2	CSN	Digital Input	SPI Chip Select
3	SCK	Digital Input	SPI Clock
4	MOSI	Digital Input	SPI Slave Data Input
5	MISO	Digital Output	SPI Slave Data Output, with tri-state option
6	IRQ	Digital Output	Maskable interrupt pin. Active low
7	VDD	Power	Power Supply (+1.9V - +3.6V DC)
8	VSS	Power	Ground (0V)
9	XC2	Analog Output	Crystal Pin 2
10	XC1	Analog Input	Crystal Pin 1
11	VDD_PA	Power Output	Power Supply Output (+1.8V) for the internal nRF24L01+ Power Amplifier. Must be connected to ANT1 and ANT2 as shown in Figure 32 .
12	ANT1	RF	Antenna interface 1
13	ANT2	RF	Antenna interface 2
14	VSS	Power	Ground (0V)
15	VDD	Power	Power Supply (+1.9V - +3.6V DC)
16	IREF	Analog Input	Reference current. Connect a 22kΩ resistor to ground. See Figure 32 .
17	VSS	Power	Ground (0V)
18	VDD	Power	Power Supply (+1.9V - +3.6V DC)
19	DVDD	Power Output	Internal digital supply output for de-coupling purposes. See Figure 32 .
20	VSS	Power	Ground (0V)

Table 1. nRF24L01+ pin function

3 Absolute maximum ratings

Note: Exceeding one or more of the limiting values may cause permanent damage to nRF24L01+.

Operating conditions	Minimum	Maximum	Units
Supply voltages			
VDD	-0.3	3.6	V
VSS		0	V
Input voltage			
V _I	-0.3	5.25	V
Output voltage			
V _O	VSS to VDD	VSS to VDD	
Total Power Dissipation			
P _D (T _A =85°C)		60	mW
Temperatures			
Operating Temperature	-40	+85	°C
Storage Temperature	-40	+125	°C

Table 2. Absolute maximum ratings

4 Operating conditions

Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
VDD	Supply voltage		1.9	3.0	3.6	V
VDD	Supply voltage if input signals >3.6V		2.7	3.0	3.3	V
TEMP	Operating Temperature		-40	+27	+85	°C

Table 3. Operating conditions

5 Electrical specifications

Conditions: $V_{DD} = +3V$, $V_{SS} = 0V$, $T_A = -40^{\circ}C$ to $+85^{\circ}C$

5.1 Power consumption

Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
Idle modes						
I_{VDD_PD}	Supply current in power down			900		nA
I_{VDD_ST1}	Supply current in standby-I mode	a		26		μA
I_{VDD_ST2}	Supply current in standby-II mode			320		μA
I_{VDD_SU}	Average current during 1.5ms crystal oscillator startup			400		μA
Transmit						
I_{VDD_TX0}	Supply current @ 0dBm output power	b		11.3		mA
I_{VDD_TX6}	Supply current @ -6dBm output power	b		9.0		mA
I_{VDD_TX12}	Supply current @ -12dBm output power	b		7.5		mA
I_{VDD_TX18}	Supply current @ -18dBm output power	b		7.0		mA
I_{VDD_AVG}	Average Supply current @ -6dBm output power, ShockBurst™	c		0.12		mA
I_{VDD_TXS}	Average current during TX settling	d		8.0		mA
Receive						
I_{VDD_2M}	Supply current 2Mbps			13.5		mA
I_{VDD_1M}	Supply current 1Mbps			13.1		mA
I_{VDD_250}	Supply current 250kbps			12.6		mA
I_{VDD_RXS}	Average current during RX settling	e		8.9		mA

- This current is for a 12pF crystal. Current when using external clock is dependent on signal swing.
- Antenna load impedance = $15\Omega + j88\Omega$.
- Antenna load impedance = $15\Omega + j88\Omega$. Average data rate 10kbps and max. payload length packets.
- Average current consumption during TX startup (130 μs) and when changing mode from RX to TX (130 μs).
- Average current consumption during RX startup (130 μs) and when changing mode from TX to RX (130 μs).

Table 4. Power consumption

5.2 General RF conditions

Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
f_{OP}	Operating frequency	a	2400		2525	MHz
PLL_{res}	PLL Programming resolution			1		MHz
f_{XTAL}	Crystal frequency			16		MHz
Δf_{250}	Frequency deviation @ 250kbps			± 160		kHz
Δf_{1M}	Frequency deviation @ 1Mbps			± 160		kHz
Δf_{2M}	Frequency deviation @ 2Mbps			± 320		kHz
R_{GFSK}	Air Data rate	b	250		2000	kbps
$F_{CHANNEL\ 1M}$	Non-overlapping channel spacing @ 250kbps/1Mbps	c		1		MHz
$F_{CHANNEL\ 2M}$	Non-overlapping channel spacing @ 2Mbps	c		2		MHz

a. Regulatory standards determine the band range you can use.

b. Data rate in each burst on-air

c. The minimum channel spacing is 1MHz

Table 5. General RF conditions

5.3 Transmitter operation

Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
P_{RF}	Maximum Output Power	a		0	+4	dBm
P_{RFC}	RF Power Control Range		16	18	20	dB
P_{RFCR}	RF Power Accuracy				± 4	dB
P_{BW2}	20dB Bandwidth for Modulated Carrier (2Mbps)			1800	2000	kHz
P_{BW1}	20dB Bandwidth for Modulated Carrier (1Mbps)			900	1000	kHz
P_{BW250}	20dB Bandwidth for Modulated Carrier (250kbps)			700	800	kHz
$P_{RF1.2}$	1 st Adjacent Channel Transmit Power 2MHz (2Mbps)				-20	dBc
$P_{RF2.2}$	2 nd Adjacent Channel Transmit Power 4MHz (2Mbps)				-50	dBc
$P_{RF1.1}$	1 st Adjacent Channel Transmit Power 1MHz (1Mbps)				-20	dBc
$P_{RF2.1}$	2 nd Adjacent Channel Transmit Power 2MHz (1Mbps)				-45	dBc
$P_{RF1.250}$	1 st Adjacent Channel Transmit Power 1MHz (250kbps)				-30	dBc
$P_{RF2.250}$	2 nd Adjacent Channel Transmit Power 2MHz (250kbps)				-45	dBc

a. Antenna load impedance = $15\Omega + j88\Omega$

Table 6. Transmitter operation

5.4 Receiver operation

Datarate	Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
	RX_{max}	Maximum received signal at <0.1% BER			0		dBm
2Mbps	RX_{SENS}	Sensitivity (0.1%BER) @2Mbps			-82		dBm
1Mbps	RX_{SENS}	Sensitivity (0.1%BER) @1Mbps			-85		dBm
250kbps	RX_{SENS}	Sensitivity (0.1%BER) @250kbps			-94		dBm

Table 7. RX Sensitivity

Datarate	Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
2Mbps	C/I_{CO}	C/I Co-channel			7		dBc
	C/I_{1ST}	1 st ACS (Adjacent Channel Selectivity) C/I 2MHz			3		dBc
	C/I_{2ND}	2 nd ACS C/I 4MHz			-17		dBc
	C/I_{3RD}	3 rd ACS C/I 6MHz			-21		dBc
	C/I_{Nth}	N th ACS C/I, $f_i > 12\text{MHz}$			-40		dBc
	C/I_{Nth}	N th ACS C/I, $f_i > 36\text{MHz}$	a		-48		dBc
1Mbps	C/I_{CO}	C/I Co-channel			9		dBc
	C/I_{1ST}	1 st ACS C/I 1MHz			8		dBc
	C/I_{2ND}	2 nd ACS C/I 2MHz			-20		dBc
	C/I_{3RD}	3 rd ACS C/I 3MHz			-30		dBc
	C/I_{Nth}	N th ACS C/I, $f_i > 6\text{MHz}$			-40		dBc
	C/I_{Nth}	N th ACS C/I, $f_i > 25\text{MHz}$	a		-47		dBc
250kbps	C/I_{CO}	C/I Co-channel			12		dBc
	C/I_{1ST}	1 st ACS C/I 1MHz			-12		dBc
	C/I_{2ND}	2 nd ACS C/I 2MHz			-33		dBc
	C/I_{3RD}	3 rd ACS C/I 3MHz			-38		dBc
	C/I_{Nth}	N th ACS C/I, $f_i > 6\text{MHz}$			-50		dBc
	C/I_{Nth}	N th ACS C/I, $f_i > 25\text{MHz}$	a		-60		dBc

a. **Narrow Band (In Band) Blocking measurements:**

0 to $\pm 40\text{MHz}$; 1MHz step size

For Interferer frequency offsets $n \cdot 2 \cdot f_{xtal}$, blocking performance is degraded by approximately 5dB compared to adjacent figures.

Table 8. RX selectivity according to ETSI EN 300 440-1 V1.3.1 (2001-09) page 27

Datarate	Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
2Mbps	C/I _{CO}	C/I Co-channel (Modulated carrier)			11		dBc
	C/I _{1ST}	1 st ACS C/I 2MHz			4		dBc
	C/I _{2ND}	2 nd ACS C/I 4MHz			-18		dBc
	C/I _{3RD}	3 rd ACS C/I 6MHz			-24		dBc
	C/I _{Nth}	N th ACS C/I, $f_i > 12\text{MHz}$			-40		dBc
	C/I _{Nth}	N th ACS C/I, $f_i > 36\text{MHz}$	a		-48		dBc
1Mbps	C/I _{CO}	C/I Co-channel			12		dBc
	C/I _{1ST}	1 st ACS C/I 1MHz			8		dBc
	C/I _{2ND}	2 nd ACS C/I 2MHz			-21		dBc
	C/I _{3RD}	3 rd ACS C/I 3MHz			-30		dBc
	C/I _{Nth}	N th ACS C/I, $f_i > 6\text{MHz}$			-40		dBc
	C/I _{Nth}	N th ACS C/I, $f_i > 25\text{MHz}$	a		-50		dBc
250kbps	C/I _{CO}	C/I Co-channel			7		dBc
	C/I _{1ST}	1 st ACS C/I 1MHz			-12		dBc
	C/I _{2ND}	2 nd ACS C/I 2MHz			-34		dBc
	C/I _{3RD}	3 rd ACS C/I 3MHz			-39		dBc
	C/I _{Nth}	N th ACS C/I, $f_i > 6\text{MHz}$			-50		dBc
	C/I _{Nth}	N th ACS C/I, $f_i > 25\text{MHz}$	a		-60		dBc

a. **Narrow Band (In Band) Blocking measurements:**

0 to $\pm 40\text{MHz}$; 1MHz step size

Wide Band Blocking measurements:

30MHz to 2000MHz; 10MHz step size

2000MHz to 2399MHz; 3MHz step size

2484MHz to 3000MHz; 3MHz step size

3GHz to 12.75GHz; 25MHz step size

Wanted signal for wideband blocking measurements:

-67dBm in 1Mbps and 2Mbps mode

-77dBm in 250kbps mode

For Interferer frequency offsets $n \cdot 2 \cdot f_{xtal}$, blocking performance are degraded by approximately 5dB compared to adjacent figures.

If the wanted signal is 3dB or more above the sensitivity level then, the carrier/interferer ratio is independent of the wanted signal level for a given frequency offset.

Table 9. RX selectivity with nRF24L01+ equal modulation on interfering signal. Measured using $P_{in} = -67\text{dBm}$ for wanted signal.

Datarate	Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
2Mbps	P_IM(6)	Input power of IM interferers at 6 and 12MHz offset from wanted signal			-42		dBm
	P_IM(8)	Input power of IM interferers at 8 and 16MHz offset from wanted signal			-38		dBm
	P_IM(10)	Input power of IM interferers at 10 and 20MHz offset from wanted signal			-37		dBm
1Mbps	P_IM(3)	Input power of IM interferers at 3 and 6MHz offset from wanted signal			-36		dBm
	P_IM(4)	Input power of IM interferers at 4 and 8MHz offset from wanted signal			-36		dBm
	P_IM(5)	Input power of IM interferers at 5 and 10MHz offset from wanted signal			-36		dBm
250kbps	P_IM(3)	Input power of IM interferers at 3 and 6MHz offset from wanted signal			-36		dBm
	P_IM(4)	Input power of IM interferers at 4 and 8MHz offset from wanted signal			-36		dBm
	P_IM(5)	Input power of IM interferers at 5 and 10MHz offset from wanted signal			-36		dBm

Note: Wanted signal level at Pin = -64 dBm. Two interferers with equal input power are used. The interferer closest in frequency is unmodulated, the other interferer is modulated equal with the wanted signal. The input power of interferers where the sensitivity equals BER = 0.1% is presented.

Table 10. RX intermodulation test performed according to Bluetooth Specification version 2.0

5.5 Crystal specifications

Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
F _{xo}	Crystal Frequency			16		MHz
ΔF	Tolerance	a b			±60	ppm
C ₀	Equivalent parallel capacitance			1.5	7.0	pF
L _s	Equivalent serial inductance	c		30		mH
C _L	Load capacitance		8	12	16	pF
ESR	Equivalent Series Resistance				100	Ω

- a. Frequency accuracy including; tolerance at 25°C, temperature drift, aging and crystal loading.
b. Frequency regulations in certain regions set tighter requirements for frequency tolerance (For example, Japan and South Korea specify max. +/- 50ppm).
c. Startup time from power down to standby mode is dependant on the L_s parameter. See [Table 16. on page 24](#) for details.

Table 11. Crystal specifications

The crystal oscillator startup time is proportional to the crystal equivalent inductance. The trend in crystal design is to reduce the physical outline. An effect of a small outline is an increase in equivalent serial inductance L_s, which gives a longer startup time. The maximum crystal oscillator startup time, T_{pd2stby} = 1.5 ms, is set using a crystal with equivalent serial inductance of maximum 30mH.

An application specific worst case startup time can be calculated as :

T_{pd2stby} = L_s/30mH * 1.5ms if L_s exceeds 30mH.

Note: In some crystal datasheets L_s is called L1 or Lm and C_s is called C1 or Cm.

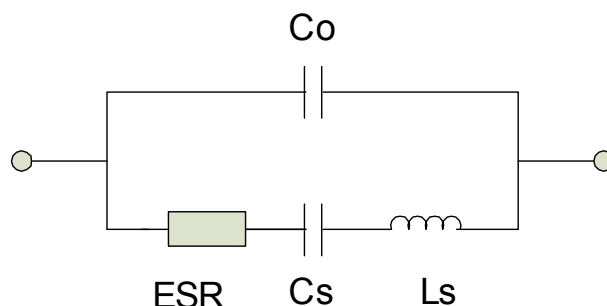


Figure 3. Equivalent crystal components

5.6 DC characteristics

Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
V_{IH}	HIGH level input voltage		$0.7V_{DD}$		5.25^a	V
V_{IL}	LOW level input voltage		V_{SS}		$0.3V_{DD}$	V

a. If the input signal $>3.6V$, the V_{DD} of the nRF24L01+ must be between 2.7V and 3.3V ($3.0V \pm 10\%$)

Table 12. Digital input pin

Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
V_{OH}	HIGH level output voltage ($I_{OH} = -0.25mA$)		$V_{DD} - 0.3$		V_{DD}	V
V_{OL}	LOW level output voltage ($I_{OL} = 0.25mA$)				0.3	V

Table 13. Digital output pin

5.7 Power on reset

Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
T_{PUP}	Power ramp up time	a			100	ms
T_{POR}	Power on reset	b	1		100	ms

a. From 0V to 1.9V.

b. Measured from when the V_{DD} reaches 1.9V to when the reset finishes.

Table 14. Power on reset

6 Radio Control

This chapter describes the nRF24L01+ radio transceiver's operating modes and the parameters used to control the radio.

The nRF24L01+ has a built-in state machine that controls the transitions between the chip's operating modes. The state machine takes input from user defined register values and internal signals.

6.1 Operational Modes

You can configure the nRF24L01+ in power down, standby, RX or TX mode. This section describes these modes in detail.

6.1.1 State diagram

The state diagram in [Figure 4](#), shows the operating modes and how they function. There are three types of distinct states highlighted in the state diagram:

- **Recommended operating mode:** is a recommended state used during normal operation.
- **Possible operating mode:** is a possible operating state, but is not used during normal operation.
- **Transition state:** is a time limited state used during start up of the oscillator and settling of the PLL.

When the V_{DD} reaches 1.9V or higher nRF24L01+ enters the Power on reset state where it remains in reset until entering the Power Down mode.

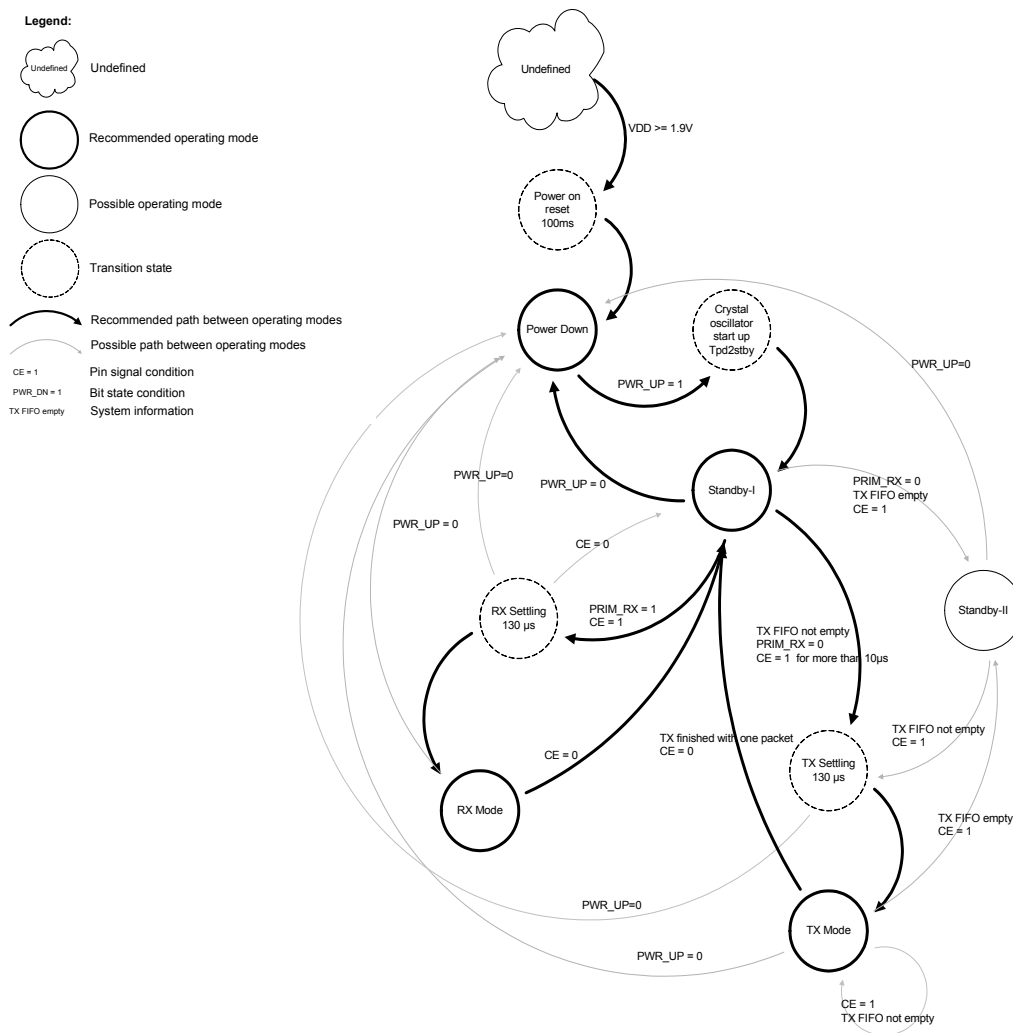


Figure 4. Radio control state diagram

6.1.3.2 Standby-II mode

In standby-II mode extra clock buffers are active and more current is used compared to standby-I mode. nRF24L01+ enters standby-II mode if **CE** is held high on a PTX device with an empty TX FIFO. If a new packet is uploaded to the TX FIFO, the PLL immediately starts and the packet is transmitted after the normal PLL settling delay (130µs).

Register values are maintained and the SPI can be activated during both standby modes. For start up times see [Table 16. on page 24](#).

6.1.4 RX mode

The RX mode is an active mode where the nRF24L01+ radio is used as a receiver. To enter this mode, the nRF24L01+ must have the **PWR_UP** bit, **PRIM_RX** bit and the **CE** pin set high.

In RX mode the receiver demodulates the signals from the RF channel, constantly presenting the demodulated data to the baseband protocol engine. The baseband protocol engine constantly searches for a valid packet. If a valid packet is found (by a matching address and a valid CRC) the payload of the packet is presented in a vacant slot in the RX FIFOs. If the RX FIFOs are full, the received packet is discarded.

The nRF24L01+ remains in RX mode until the MCU configures it to standby-I mode or power down mode. However, if the automatic protocol features (Enhanced ShockBurst™) in the baseband protocol engine are enabled, the nRF24L01+ can enter other modes in order to execute the protocol.

In RX mode a Received Power Detector (RPD) signal is available. The RPD is a signal that is set high when a RF signal higher than -64 dBm is detected inside the receiving frequency channel. The internal **RPD** signal is filtered before presented to the **RPD** register. The RF signal must be present for at least 40µs before the **RPD** is set high. How to use the RPD is described in [Section 6.4 on page 25](#).

6.1.5 TX mode

The TX mode is an active mode for transmitting packets. To enter this mode, the nRF24L01+ must have the **PWR_UP** bit set high, **PRIM_RX** bit set low, a payload in the TX FIFO and a high pulse on the **CE** for more than 10µs.

The nRF24L01+ stays in TX mode until it finishes transmitting a packet. If **CE** = 0, nRF24L01+ returns to standby-I mode. If **CE** = 1, the status of the TX FIFO determines the next action. If the TX FIFO is not empty the nRF24L01+ remains in TX mode and transmits the next packet. If the TX FIFO is empty the nRF24L01+ goes into standby-II mode. The nRF24L01+ transmitter PLL operates in open loop when in TX mode. It is important never to keep the nRF24L01+ in TX mode for more than 4ms at a time. If the Enhanced ShockBurst™ features are enabled, nRF24L01+ is never in TX mode longer than 4ms.

6.1.6 Operational modes configuration

The following table ([Table 15.](#)) describes how to configure the operational modes.

Mode	PWR_UP register	PRIM_RX register	CE input pin	FIFO state
RX mode	1	1	1	-
TX mode	1	0	1	Data in TX FIFOs. Will empty all levels in TX FIFOs ^a .
TX mode	1	0	Minimum 10µs high pulse	Data in TX FIFOs. Will empty one level in TX FIFOs ^b .
Standby-II	1	0	1	TX FIFO empty.
Standby-I	1	-	0	No ongoing packet transmission.
Power Down	0	-	-	-

- If **CE** is held high all TX FIFOs are emptied and all necessary ACK and possible retransmits are carried out. The transmission continues as long as the TX FIFO is refilled. If the TX FIFO is empty when the **CE** is still high, nRF24L01+ enters standby-II mode. In this mode the transmission of a packet is started as soon as the **CSN** is set high after an upload (UL) of a packet to TX FIFO.
- This operating mode pulses the **CE** high for at least 10µs. This allows one packet to be transmitted. This is the normal operating mode. After the packet is transmitted, the nRF24L01+ enters standby-I mode.

Table 15. nRF24L01+ main modes

6.1.7 Timing Information

The timing information in this section relates to the transitions between modes and the timing for the **CE** pin. The transition from TX mode to RX mode or vice versa is the same as the transition from the standby modes to TX mode or RX mode (max. 130µs), as described in [Table 16.](#)

Name	nRF24L01+	Notes	Max.	Min.	Comments
Tpd2stby	Power Down → Standby mode	a	150µs		With external clock
			1.5ms		External crystal, Ls < 30mH
			3ms		External crystal, Ls = 60mH
			4.5ms		External crystal, Ls = 90mH
Tstby2a	Standby modes → TX/RX mode		130µs		
Thce	Minimum CE high			10µs	
Tpece2csn	Delay from CE positive edge to CSN low			4µs	

- See [Table 11. on page 19](#) for crystal specifications.

Table 16. Operational timing of nRF24L01+

For nRF24L01+ to go from power down mode to TX or RX mode it must first pass through stand-by mode. There must be a delay of Tpd2stby (see [Table 16.](#)) after the nRF24L01+ leaves power down mode before the **CE** is set high.

Note: If **VDD** is turned off the register value is lost and you must configure nRF24L01+ before entering the TX or RX modes.

6.2 Air data rate

The air data rate is the modulated signaling rate the nRF24L01+ uses when transmitting and receiving data. It can be 250kbps, 1Mbps or 2Mbps. Using lower air data rate gives better receiver sensitivity than higher air data rate. But, high air data rate gives lower average current consumption and reduced probability of on-air collisions.

The air data rate is set by the `RF_DR` bit in the `RF_SETUP` register. A transmitter and a receiver must be programmed with the same air data rate to communicate with each other.

nRF24L01+ is fully compatible with nRF24L01. For compatibility with nRF2401A, nRF2402, nRF24E1, and nRF24E2 the air data rate must be set to 250kbps or 1Mbps.

6.3 RF channel frequency

The RF channel frequency determines the center of the channel used by the nRF24L01+. The channel occupies a bandwidth of less than 1MHz at 250kbps and 1Mbps and a bandwidth of less than 2MHz at 2Mbps. nRF24L01+ can operate on frequencies from 2.400GHz to 2.525GHz. The programming resolution of the RF channel frequency setting is 1MHz.

At 2Mbps the channel occupies a bandwidth wider than the resolution of the RF channel frequency setting. To ensure non-overlapping channels in 2Mbps mode, the channel spacing must be 2MHz or more. At 1Mbps and 250kbps the channel bandwidth is the same or lower than the resolution of the RF frequency.

The RF channel frequency is set by the `RF_CH` register according to the following formula:

$$F_0 = 2400 + RF_CH [MHz]$$

You must program a transmitter and a receiver with the same RF channel frequency to communicate with each other.

6.4 Received Power Detector measurements

Received Power Detector (RPD), located in register 09, bit 0, triggers at received power levels above -64 dBm that are present in the RF channel you receive on. If the received power is less than -64 dBm, RDP = 0.

The RPD can be read out at any time while nRF24L01+ is in receive mode. This offers a snapshot of the current received power level in the channel. The RPD status is latched when a valid packet is received which then indicates signal strength from your own transmitter. If no packets are received the RPD is latched at the end of a receive period as a result of host MCU setting CE low or RX time out controlled by Enhanced ShockBurst™.

The status of RPD is correct when RX mode is enabled and after a wait time of $T_{stby2a} + T_{delay_AGC} = 130\mu s + 40\mu s$. The RX gain varies over temperature which means that the RPD threshold also varies over temperature. The RPD threshold value is reduced by - 5dB at $T = -40^{\circ}C$ and increased by + 5dB at $85^{\circ}C$.

6.5 PA control

The PA (Power Amplifier) control is used to set the output power from the nRF24L01+ power amplifier. In TX mode PA control has four programmable steps, see [Table 17](#).

The PA control is set by the `RF_PWR` bits in the `RF_SETUP` register.

SPI RF-SETUP (RF_PWR)	RF output power	DC current consumption
11	0dBm	11.3mA
10	-6dBm	9.0mA
01	-12dBm	7.5mA
00	-18dBm	7.0mA

Conditions: $V_{DD} = 3.0V$, $V_{SS} = 0V$, $T_A = 27^{\circ}C$, Load impedance = $15\Omega + j88\Omega$.

Table 17. RF output power setting for the nRF24L01+

6.6 RX/TX control

The RX/TX control is set by `PRIM_RX` bit in the `CONFIG` register and sets the nRF24L01+ in transmit/receive mode.

7 Enhanced ShockBurst™

Enhanced ShockBurst™ is a packet based data link layer that features automatic packet assembly and timing, automatic acknowledgement and retransmissions of packets. Enhanced ShockBurst™ enables the implementation of ultra low power and high performance communication with low cost host microcontrollers. The Enhanced ShockBurst™ features enable significant improvements of power efficiency for bi-directional and uni-directional systems, without adding complexity on the host controller side.

7.1 Features

The main features of Enhanced ShockBurst™ are:

- 1 to 32 bytes dynamic payload length
- Automatic packet handling
- Automatic packet transaction handling
 - Auto Acknowledgement with payload
 - Auto retransmit
- 6 data pipe MultiCeiver™ for 1:6 star networks

7.2 Enhanced ShockBurst™ overview

Enhanced ShockBurst™ uses ShockBurst™ for automatic packet handling and timing. During transmit, ShockBurst™ assembles the packet and clocks the bits in the data packet for transmission. During receive, ShockBurst™ constantly searches for a valid address in the demodulated signal. When ShockBurst™ finds a valid address, it processes the rest of the packet and validates it by CRC. If the packet is valid the payload is moved into a vacant slot in the RX FIFOs. All high speed bit handling and timing is controlled by ShockBurst™.

Enhanced ShockBurst™ features automatic packet transaction handling for the easy implementation of a reliable bi-directional data link. An Enhanced ShockBurst™ packet transaction is a packet exchange between two transceivers, with one transceiver acting as the Primary Receiver (PRX) and the other transceiver acting as the Primary Transmitter (PTX). An Enhanced ShockBurst™ packet transaction is always initiated by a packet transmission from the PTX, the transaction is complete when the PTX has received an acknowledgment packet (ACK packet) from the PRX. The PRX can attach user data to the ACK packet enabling a bi-directional data link.

The automatic packet transaction handling works as follows:

1. You begin the transaction by transmitting a data packet from the PTX to the PRX. Enhanced ShockBurst™ automatically sets the PTX in receive mode to wait for the ACK packet.
2. If the packet is received by the PRX, Enhanced ShockBurst™ automatically assembles and transmits an acknowledgment packet (ACK packet) to the PTX before returning to receive mode.
3. If the PTX does not receive the ACK packet immediately, Enhanced ShockBurst™ automatically retransmits the original data packet after a programmable delay and sets the PTX in receive mode to wait for the ACK packet.

In Enhanced ShockBurst™ it is possible to configure parameters such as the maximum number of retransmits and the delay from one transmission to the next retransmission. All automatic handling is done without the involvement of the MCU.

7.3 Enhanced Shockburst™ packet format

The format of the Enhanced ShockBurst™ packet is described in this section. The Enhanced ShockBurst™ packet contains a preamble, address, packet control, payload and CRC field. [Figure 5.](#) shows the packet format with MSB to the left.



Figure 5. An Enhanced ShockBurst™ packet with payload (0-32 bytes)

7.3.1 Preamble

The preamble is a bit sequence used to synchronize the receivers demodulator to the incoming bit stream. The preamble is one byte long and is either 01010101 or 10101010. If the first bit in the address is 1 the preamble is automatically set to 10101010 and if the first bit is 0 the preamble is automatically set to 01010101. This is done to ensure there are enough transitions in the preamble to stabilize the receiver.

7.3.2 Address

This is the address for the receiver. An address ensures that the packet is detected and received by the correct receiver, preventing accidental cross talk between multiple nRF24L01+ systems. You can configure the address field width in the `AW` register to be 3, 4 or 5 bytes, see [Table 28. on page 63](#).

Note: Addresses where the level shifts only one time (that is, 000FFFFFFF) can often be detected in noise and can give a false detection, which may give a raised Packet Error Rate. Addresses as a continuation of the preamble (hi-low toggling) also raises the Packet Error Rate.

7.3.3 Packet control field

[Figure 6.](#) shows the format of the 9 bit packet control field, MSB to the left.

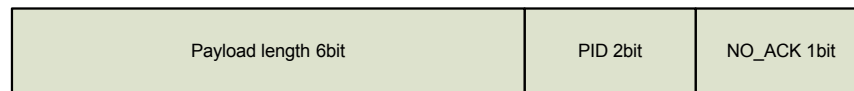


Figure 6. Packet control field

The packet control field contains a 6 bit payload length field, a 2 bit PID (Packet Identity) field and a 1 bit `NO_ACK` flag.

7.3.3.1 Payload length

This 6 bit field specifies the length of the payload in bytes. The length of the payload can be from 0 to 32 bytes.

Coding: 000000 = 0 byte (only used in empty ACK packets.) 100000 = 32 byte, 100001 = Don't care.

This field is only used if the Dynamic Payload Length function is enabled.

7.3.3.2 PID (Packet identification)

The 2 bit PID field is used to detect if the received packet is new or retransmitted. PID prevents the PRX device from presenting the same payload more than once to the receiving host MCU. The PID field is incremented at the TX side for each new packet received through the SPI. The PID and CRC fields (see [section 7.3.5 on page 30](#)) are used by the PRX device to determine if a packet is retransmitted or new. When several data packets are lost on the link, the PID fields may become equal to the last received PID. If a packet has the same PID as the previous packet, nRF24L01+ compares the CRC sums from both packets. If the CRC sums are also equal, the last received packet is considered a copy of the previously received packet and discarded.

7.3.3.3 No Acknowledgment flag (NO_ACK)

The Selective Auto Acknowledgement feature controls the NO_ACK flag.

This flag is only used when the auto acknowledgement feature is used. Setting the flag high tells the receiver that the packet is not to be auto acknowledged.

On the PTX you can set the NO_ACK flag bit in the Packet Control Field with this command:

```
W_TX_PAYLOAD_NOACK
```

However, the function must first be enabled in the FEATURE register by setting the EN_DYN_ACK bit. When you use this option the PTX goes directly to standby-I mode after transmitting the packet. The PRX does not transmit an ACK packet when it receives the packet.

7.3.4 Payload

The payload is the user defined content of the packet. It can be 0 to 32 bytes wide and is transmitted on-air when it is uploaded to nRF24L01+.

Enhanced ShockBurst™ provides two alternatives for handling payload lengths; static and dynamic.

The default is static payload length. With static payload length all packets between a transmitter and a receiver have the same length. Static payload length is set by the RX_PW_Px registers on the receiver side. The payload length on the transmitter side is set by the number of bytes clocked into the TX_FIFO and must equal the value in the RX_PW_Px register on the receiver side.

Dynamic Payload Length (DPL) is an alternative to static payload length. DPL enables the transmitter to send packets with variable payload length to the receiver. This means that for a system with different payload lengths it is not necessary to scale the packet length to the longest payload.

With the DPL feature the nRF24L01+ can decode the payload length of the received packet automatically instead of using the `RX_PW_Px` registers. The MCU can read the length of the received payload by using the `R_RX_PL_WID` command.

Note: Always check if the packet width reported is 32 bytes or shorter when using the `R_RX_PL_WID` command. If its width is longer than 32 bytes then the packet contains errors and must be discarded. Discard the packet by using the `Flush_RX` command.

In order to enable DPL the `EN_DPL` bit in the `FEATURE` register must be enabled. In RX mode the `DYNPD` register must be set. A PTX that transmits to a PRX with DPL enabled must have the `DPL_P0` bit in `DYNPD` set.

7.3.5 CRC (Cyclic Redundancy Check)

The CRC is the mandatory error detection mechanism in the packet. It is either 1 or 2 bytes and is calculated over the address, Packet Control Field and Payload.

The polynomial for 1 byte CRC is $X^8 + X^2 + X + 1$. Initial value 0xFF.

The polynomial for 2 byte CRC is $X^{16} + X^{12} + X^5 + 1$. Initial value 0xFFFF.

The number of bytes in the CRC is set by the `CRCO` bit in the `CONFIG` register. No packet is accepted by Enhanced ShockBurst™ if the CRC fails.

7.3.6 Automatic packet assembly

The automatic packet assembly assembles the preamble, address, packet control field, payload and CRC to make a complete packet before it is transmitted.

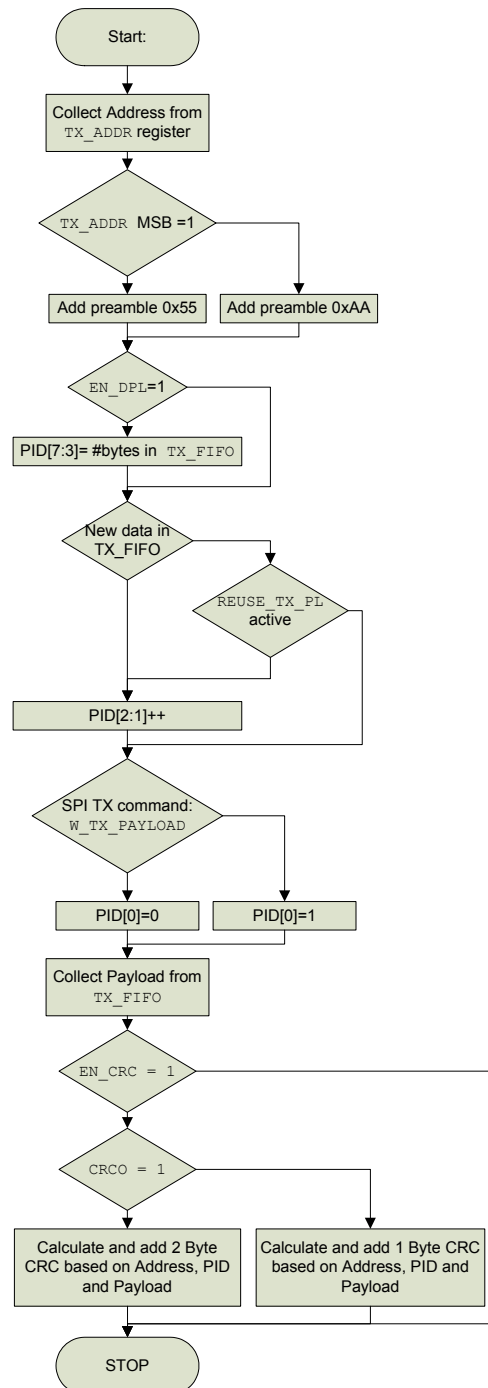


Figure 7. Automatic packet assembly

7.3.7 Automatic packet disassembly

After the packet is validated, Enhanced ShockBurst™ disassembles the packet and loads the payload into the RX FIFO, and asserts the `RX_DR` IRQ.

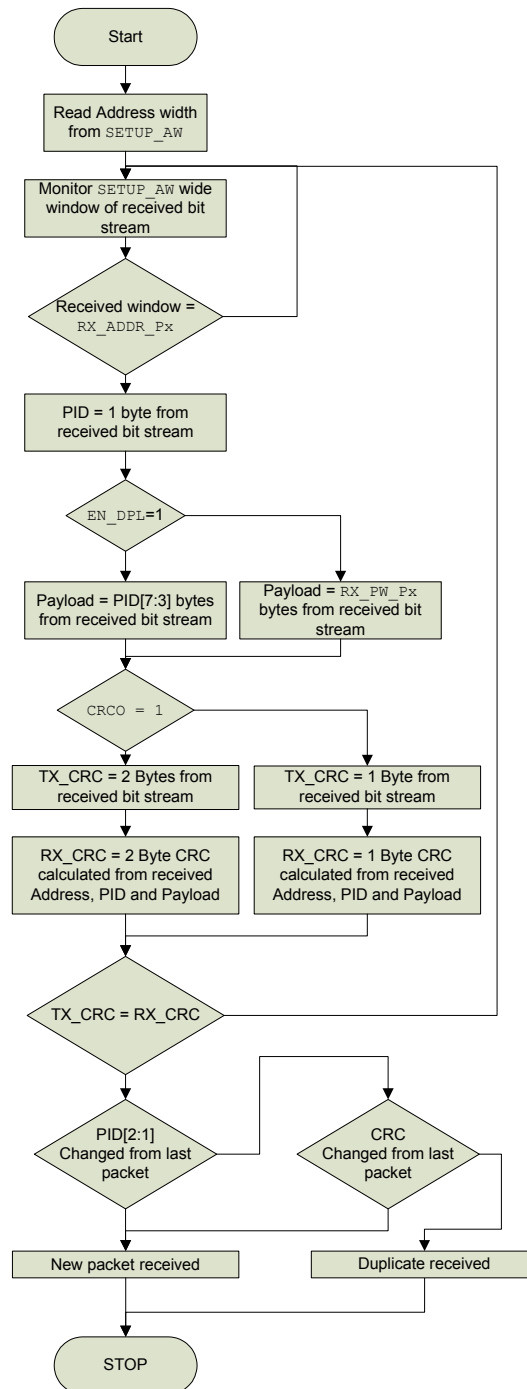


Figure 8. Automatic packet disassembly

7.4 Automatic packet transaction handling

Enhanced ShockBurst™ has two functions for automatic packet transaction handling; auto acknowledgement and auto re-transmit.

7.4.1 Auto acknowledgement

Auto acknowledgement is a function that automatically transmits an ACK packet to the PTX after it has received and validated a packet. The auto acknowledgement function reduces the load of the system MCU and can remove the need for dedicated SPI hardware. This also reduces cost and average current consumption. The Auto Acknowledgement feature is enabled by setting the `EN_AA` register.

Note: If the received packet has the `NO_ACK` flag set, auto acknowledgement is not executed.

An ACK packet can contain an optional payload from PRX to PTX. In order to use this feature, the Dynamic Payload Length (DPL) feature must be enabled. The MCU on the PRX side has to upload the payload by clocking it into the TX FIFO by using the `W_ACK_PAYLOAD` command. The payload is pending in the TX FIFO (PRX) until a new packet is received from the PTX. nRF24L01+ can have three ACK packet payloads pending in the TX FIFO (PRX) at the same time.

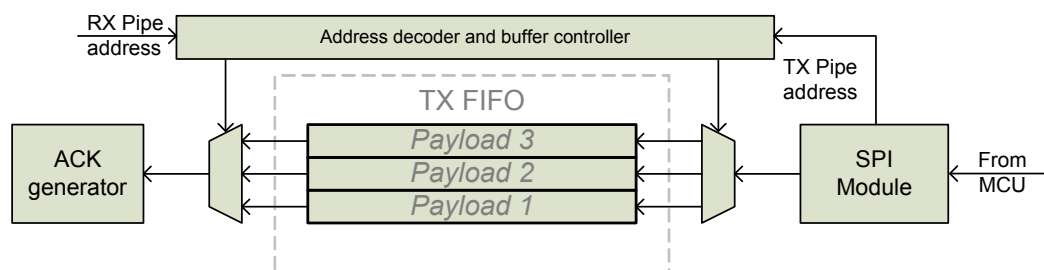


Figure 9. TX FIFO (PRX) with pending payloads

Figure 9. shows how the TX FIFO (PRX) is operated when handling pending ACK packet payloads. From the MCU the payload is clocked in with the `W_ACK_PAYLOAD` command. The address decoder and buffer controller ensure that the payload is stored in a vacant slot in the TX FIFO (PRX). When a packet is received, the address decoder and buffer controller are notified with the PTX address. This ensures that the right payload is presented to the ACK generator.

If the TX FIFO (PRX) contains more than one payload to a PTX, payloads are handled using the first in – first out principle. The TX FIFO (PRX) is blocked if all pending payloads are addressed to a PTX where the link is lost. In this case, the MCU can flush the TX FIFO (PRX) by using the `FLUSH_TX` command.

In order to enable Auto Acknowledgement with payload the `EN_ACK_PAY` bit in the `FEATURE` register must be set.

7.4.2 Auto Retransmission (ART)

The auto retransmission is a function that retransmits a packet if an ACK packet is not received. It is used in an auto acknowledgement system on the PTX. When a packet is not acknowledged, you can set the number of times it is allowed to retransmit by setting the ARC bits in the `SETUP_RETR` register. PTX enters RX mode and waits a short period for an ACK packet each time a packet is transmitted. The time period the PTX is in RX mode is based on the following conditions:

- Auto Retransmit Delay (ARD) has elapsed.
- No address match within 250µs (or 500µs in 250kbps mode).
- After received packet (CRC correct or not).

nRF24L01+ asserts the `TX_DS` IRQ when the ACK packet is received.

nRF24L01+ enters standby-I mode if there is no more untransmitted data in the TX FIFO and the `CE` pin is low. If the ACK packet is not received, nRF24L01+ goes back to TX mode after a delay defined by ARD and retransmits the data. This continues until acknowledgment is received, or the maximum number of retransmits is reached.

Two packet loss counters are incremented each time a packet is lost, `ARC_CNT` and `PLOS_CNT` in the `OBSERVE_TX` register. The `ARC_CNT` counts the number of retransmissions for the current transaction. You reset `ARC_CNT` by initiating a new transaction. The `PLOS_CNT` counts the total number of retransmissions since the last channel change. You reset `PLOS_CNT` by writing to the `RF_CH` register. It is possible to use the information in the `OBSERVE_TX` register to make an overall assessment of the channel quality.

The ARD defines the time from the end of a transmitted packet to when a retransmit starts on the PTX. ARD is set in `SETUP_RETR` register in steps of 250µs. A retransmit is made if no ACK packet is received by the PTX.

There is a restriction on the length of ARD when using ACK packets with payload. The ARD time must never be shorter than the sum of the startup time and the time on-air for the ACK packet:

- For 2Mbps data rate and 5 byte address; 15 byte is maximum ACK packet payload length for ARD=250µs (reset value).
- For 1Mbps data rate and 5 byte address; 5 byte is maximum ACK packet payload length for ARD=250µs (reset value).

ARD=500µs is long enough for any ACK payload length in 1 or 2Mbps mode.

- For 250kbps data rate and 5byte address the following values apply:

ARD	ACK packet size (in bytes)
1500µs	All ACK payload sizes
1250µs	≤ 24
1000µs	≤ 16
750µs	≤ 8
500µs	Empty ACK with no payload

Table 18. Maximum ACK payload length for different retransmit delays at 250kbps

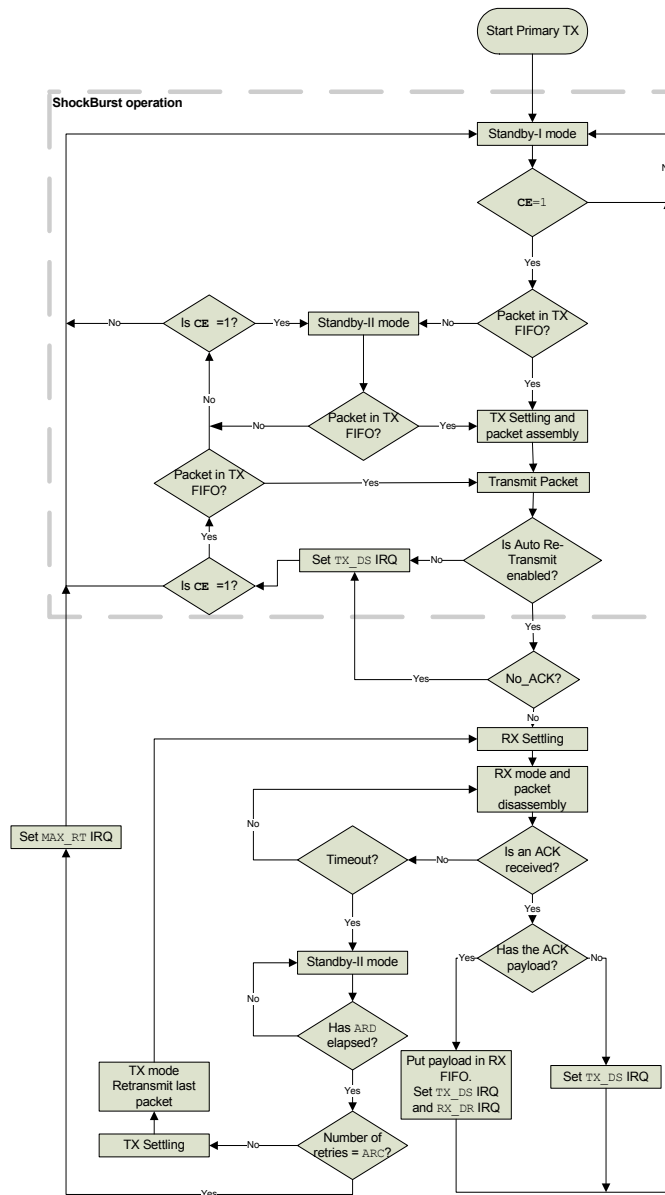
As an alternative to Auto Retransmit it is possible to manually set the nRF24L01+ to retransmit a packet a number of times. This is done by the `REUSE_TX_PL` command. The MCU must initiate each transmission of the packet with a pulse on the `CE` pin when this command is used.

7.5 Enhanced ShockBurst flowcharts

This section contains flowcharts outlining PTX and PRX operation in Enhanced ShockBurst™.

7.5.1 PTX operation

The flowchart in [Figure 10](#) outlines how a nRF24L01+ configured as a PTX behaves after entering standby-I mode.



Note: ShockBurst™ operation is outlined with a dashed square.

Figure 10. PTX operations in Enhanced ShockBurst™

Activate PTX mode by setting the \overline{CE} pin high. If there is a packet present in the TX FIFO the nRF24L01+ enters TX mode and transmits the packet. If Auto Retransmit is enabled, the state machine checks if the NO_ACK flag is set. If it is not set, the nRF24L01+ enters RX mode to receive an ACK packet. If the received ACK packet is empty, only the TX_DS IRQ is asserted. If the ACK packet contains a payload, both TX_DS IRQ and RX_DR IRQ are asserted simultaneously before nRF24L01+ returns to standby-I mode.

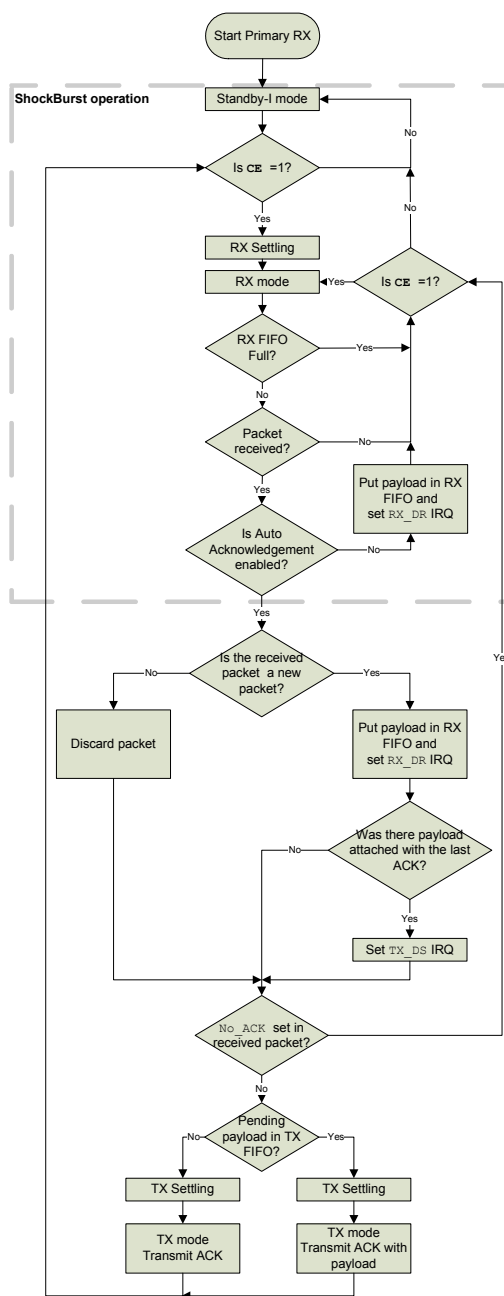
If the ACK packet is not received before timeout occurs, the nRF24L01+ returns to standby-II mode. It stays in standby-II mode until the ARD has elapsed. If the number of retransmits has not reached the ARC, the nRF24L01+ enters TX mode and transmits the last packet once more.

While executing the Auto Retransmit feature, the number of retransmits can reach the maximum number defined in ARC. If this happens, the nRF24L01+ asserts the MAX_RT IRQ and returns to standby-I mode.

If the \overline{CE} is high and the TX FIFO is empty, the nRF24L01+ enters Standby-II mode.

7.5.2 PRX operation

The flowchart in [Figure 11](#) outlines how a nRF24L01+ configured as a PRX behaves after entering standby-I mode.



Note: ShockBurst™ operation is outlined with a dashed square.

Figure 11. PRX operations in Enhanced ShockBurst™

Activate PRX mode by setting the **CE** pin high. The nRF24L01+ enters RX mode and starts searching for packets. If a packet is received and Auto Acknowledgement is enabled, nRF24L01+ decides if the packet is new or a copy of a previously received packet. If the packet is new the payload is made available in the

RX FIFO and the `RX_DR` IRQ is asserted. If the last received packet from the transmitter is acknowledged with an ACK packet with payload, the `TX_DS` IRQ indicates that the PTX received the ACK packet with payload. If the `NO_ACK` flag is not set in the received packet, the PRX enters TX mode. If there is a pending payload in the TX FIFO it is attached to the ACK packet. After the ACK packet is transmitted, the nRF24L01+ returns to RX mode.

A copy of a previously received packet might be received if the ACK packet is lost. In this case, the PRX discards the received packet and transmits an ACK packet before it returns to RX mode.

7.6 MultiCeiver™

MultiCeiver™ is a feature used in RX mode that contains a set of six parallel data pipes with unique addresses. A data pipe is a logical channel in the physical RF channel. Each data pipe has its own physical address (data pipe address) decoding in the nRF24L01+.

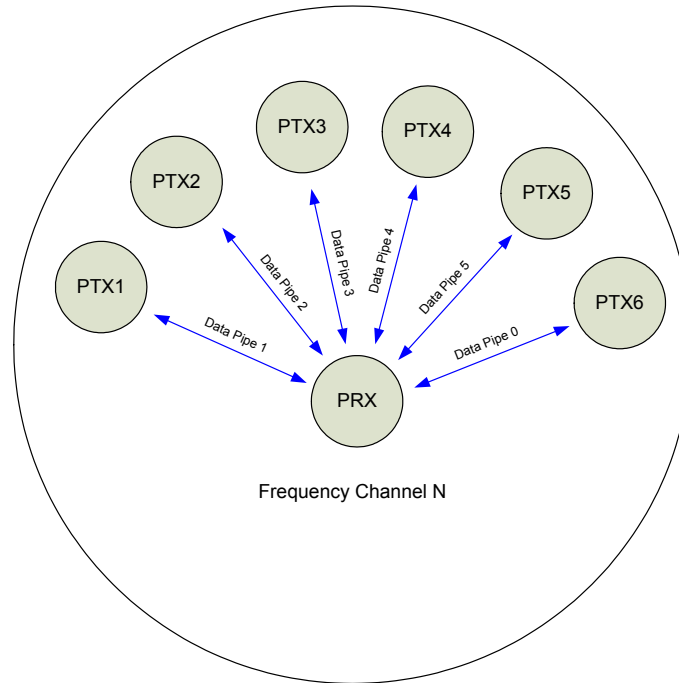


Figure 12. PRX using MultiCeiver™

nRF24L01+ configured as PRX (primary receiver) can receive data addressed to six different data pipes in one frequency channel as shown in [Figure 12](#). Each data pipe has its own unique address and can be configured for individual behavior.

Up to six nRF24L01+s configured as PTX can communicate with one nRF24L01+ configured as a PRX. All data pipe addresses are searched for simultaneously. Only one data pipe can receive a packet at a time. All data pipes can perform Enhanced ShockBurst™ functionality.

The following settings are common to all data pipes:

- CRC enabled/disabled (CRC always enabled when Enhanced ShockBurst™ is enabled)
- CRC encoding scheme
- RX address width
- Frequency channel
- Air data rate
- LNA gain

The data pipes are enabled with the bits in the `EN_RXADDR` register. By default only data pipe 0 and 1 are enabled. Each data pipe address is configured in the `RX_ADDR_PX` registers.

Note: Always ensure that none of the data pipes have the same address.

Each pipe can have up to a 5 byte configurable address. Data pipe 0 has a unique 5 byte address. Data pipes 1-5 share the four most significant address bytes. The LSByte must be unique for all six pipes. [Figure 13](#) is an example of how data pipes 0-5 are addressed.

	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
Data pipe 0 (RX_ADDR_P0)	0xE7	0xD3	0xF0	0x35	0x77
Data pipe 1 (RX_ADDR_P1)	0xC2	0xC2	0xC2	0xC2	0xC2
	↓	↓	↓	↓	
Data pipe 2 (RX_ADDR_P2)	0xC2	0xC2	0xC2	0xC2	0xC3
	↓	↓	↓	↓	
Data pipe 3 (RX_ADDR_P3)	0xC2	0xC2	0xC2	0xC2	0xC4
	↓	↓	↓	↓	
Data pipe 4 (RX_ADDR_P4)	0xC2	0xC2	0xC2	0xC2	0xC5
	↓	↓	↓	↓	
Data pipe 5 (RX_ADDR_P5)	0xC2	0xC2	0xC2	0xC2	0xC6

Figure 13. Addressing data pipes 0-5

The PRX, using MultiCeiver™ and Enhanced ShockBurst™, receives packets from more than one PTX. To ensure that the ACK packet from the PRX is transmitted to the correct PTX, the PRX takes the data pipe address where it received the packet and uses it as the TX address when transmitting the ACK packet. [Figure 14](#) is an example of an address configuration for the PRX and PTX. On the PRX the RX_ADDR_Pn, defined as the pipe address, must be unique. On the PTX the TX_ADDR must be the same as the RX_ADDR_P0 and as the pipe address for the designated pipe.

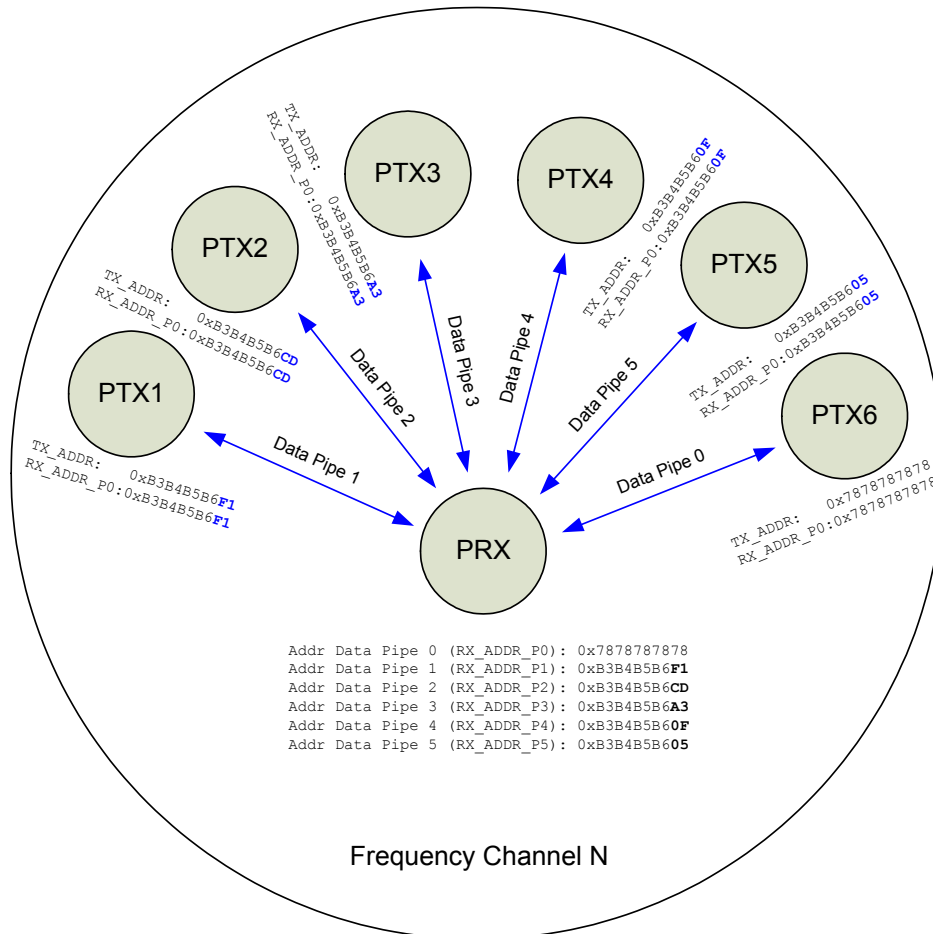


Figure 14. Example of data pipe addressing in MultiCeiver™

Only when a data pipe receives a complete packet can other data pipes begin to receive data. When multiple PTXs are transmitting to a PRX, the ARD can be used to skew the auto retransmission so that they only block each other once.

7.7 Enhanced ShockBurst™ timing

This section describes the timing sequence of Enhanced ShockBurst™ and how all modes are initiated and operated. The Enhanced ShockBurst™ timing is controlled through the Data and Control interface. The nRF24L01+ can be set to static modes or autonomous modes where the internal state machine controls the events. Each autonomous mode/sequence ends with an interrupt at the **IRQ** pin. All the interrupts are indicated as IRQ events in the timing diagrams.

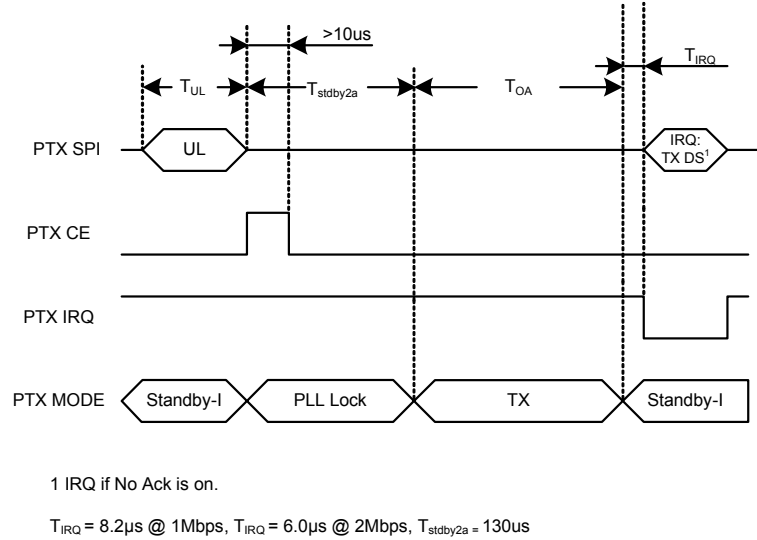


Figure 15. Transmitting one packet with NO_ACK on

The following equations calculate various timing measurements:

Symbol	Description	Equation
T_{OA}	Time on-air	$T_{\text{OA}} = \frac{\text{packet length}}{\text{air data rate}} = \frac{8 \left[\frac{\text{bit}}{\text{byte}} \right] \cdot \left(1 \left[\frac{\text{byte}}{\text{preamble}} \right] + 3, 4 \text{ or } 5 \left[\frac{\text{bytes}}{\text{address}} \right] + N \left[\frac{\text{bytes}}{\text{payload}} \right] + 1 \text{ or } 2 \left[\frac{\text{bytes}}{\text{CRC}} \right] \right) + 9 \left[\frac{\text{bit}}{\text{packet control field}} \right]}{\text{air data rate} \left[\frac{\text{bit}}{\text{s}} \right]}$
T_{ACK}	Time on-air Ack	$T_{\text{ACK}} = \frac{\text{packet length}}{\text{air data rate}} = \frac{8 \left[\frac{\text{bit}}{\text{byte}} \right] \cdot \left(1 \left[\frac{\text{byte}}{\text{preamble}} \right] + 3, 4 \text{ or } 5 \left[\frac{\text{bytes}}{\text{address}} \right] + N \left[\frac{\text{bytes}}{\text{payload}} \right] + 1 \text{ or } 2 \left[\frac{\text{bytes}}{\text{CRC}} \right] \right) + 9 \left[\frac{\text{bit}}{\text{packet control field}} \right]}{\text{air data rate} \left[\frac{\text{bit}}{\text{s}} \right]}$
T_{UL}	Time Upload	$T_{\text{UL}} = \frac{\text{payload length}}{\text{SPI data rate}} = \frac{8 \left[\frac{\text{bit}}{\text{byte}} \right] \cdot N \left[\frac{\text{bytes}}{\text{payload}} \right]}{\text{SPI data rate} \left[\frac{\text{bit}}{\text{s}} \right]}$
T_{ESB}	Time Enhanced ShockBurst™ cycle	$T_{\text{ESB}} = T_{\text{UL}} + 2 \cdot T_{\text{stby2a}} + T_{\text{OA}} + T_{\text{ACK}} + T_{\text{IRQ}}$

Table 19. Timing equations

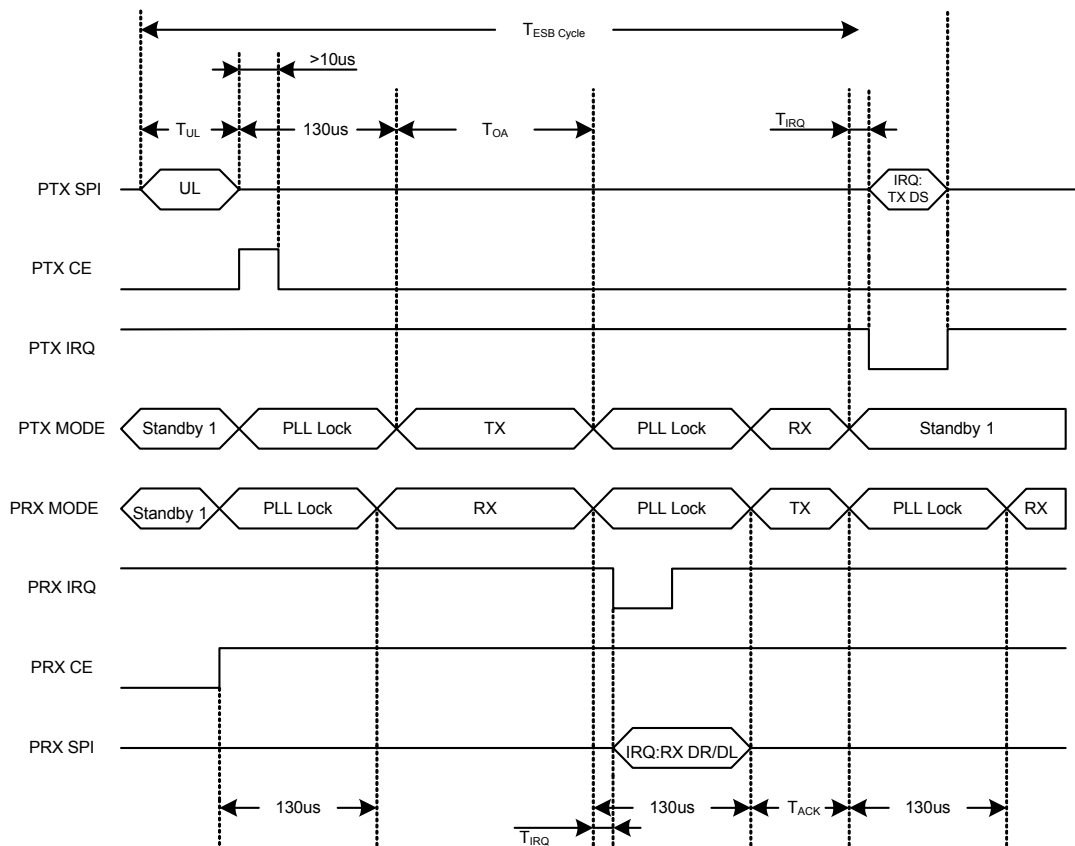


Figure 16. Timing of Enhanced ShockBurst™ for one packet upload (2Mbps)

In [Figure 16](#), the transmission and acknowledgement of a packet is shown. The PRX device activates RX mode ($CE=1$), and the PTX device is activated in TX mode ($CE=1$ for minimum 10μs). After 130μs the transmission starts and finishes after the elapse of T_{OA} .

When the transmission ends the PTX device automatically switches to RX mode to wait for the ACK packet from the PRX device. When the PRX device receives the packet it sets the interrupt for the host MCU and switches to TX mode to send an ACK. After the PTX device receives the ACK packet it sets the interrupt to the MCU and clears the packet from the TX FIFO.

In [Figure 17](#), the PTX timing of a packet transmission is shown when the first ACK packet is lost. To see the complete transmission when the ACK packet fails see [Figure 20. on page 46](#).

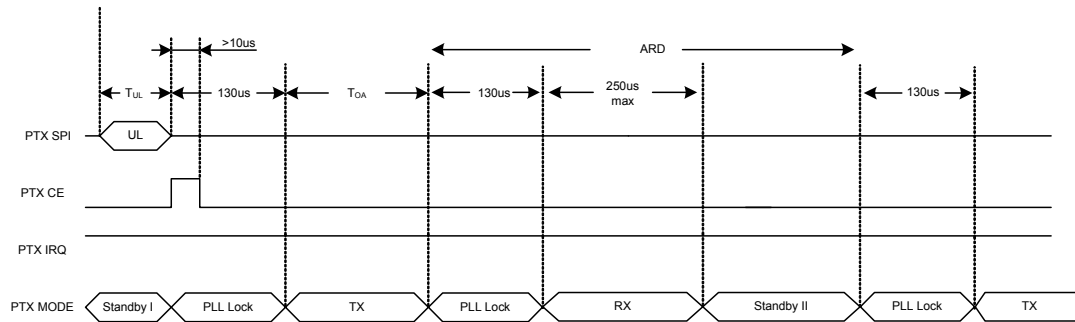


Figure 17. Timing of Enhanced ShockBurst™ when the first ACK packet is lost (2Mbps)

7.8 Enhanced ShockBurst™ transaction diagram

This section describes several scenarios for the Enhanced ShockBurst™ automatic transaction handling. The call outs in this section's figures indicate the IRQs and other events. For MCU activity the event may be placed at a different timeframe.

Note: The figures in this section indicate the earliest possible download (DL) of the packet to the MCU and the latest possible upload (UL) of payload to the transmitter.

7.8.1 Single transaction with ACK packet and interrupts

In [Figure 18](#), the basic auto acknowledgement is shown. After the packet is transmitted by the PTX and received by the PRX the ACK packet is transmitted from the PRX to the PTX. The `RX_DR` IRQ is asserted after the packet is received by the PRX, whereas the `TX_DS` IRQ is asserted when the packet is acknowledged and the ACK packet is received by the PTX.

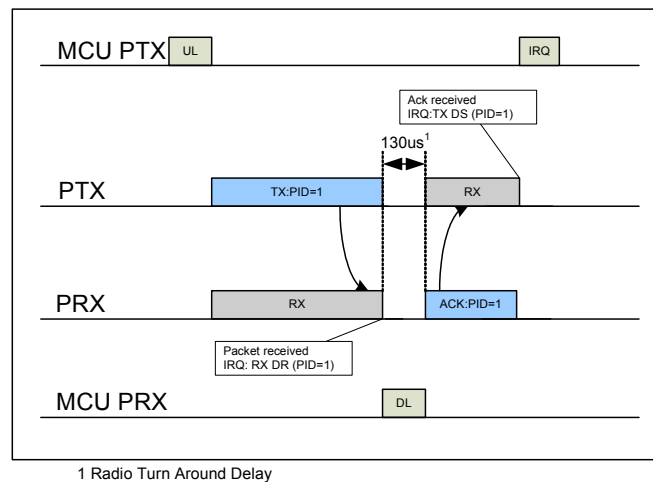


Figure 18. TX/RX cycles with ACK and the according interrupts

7.8.4 Single transaction with ACK payload packet

Figure 21. is a scenario of the basic auto acknowledgement with payload. After the packet is transmitted by the PTX and received by the PRX the ACK packet with payload is transmitted from the PRX to the PTX. The `RX_DR` IRQ is asserted after the packet is received by the PRX, whereas on the PTX side the `TX_DS` IRQ is asserted when the ACK packet is received by the PTX. On the PRX side, the `TX_DS` IRQ for the ACK packet payload is asserted after a new packet from PTX is received. The position of the IRQ in Figure 21. shows where the MCU can respond to the interrupt.

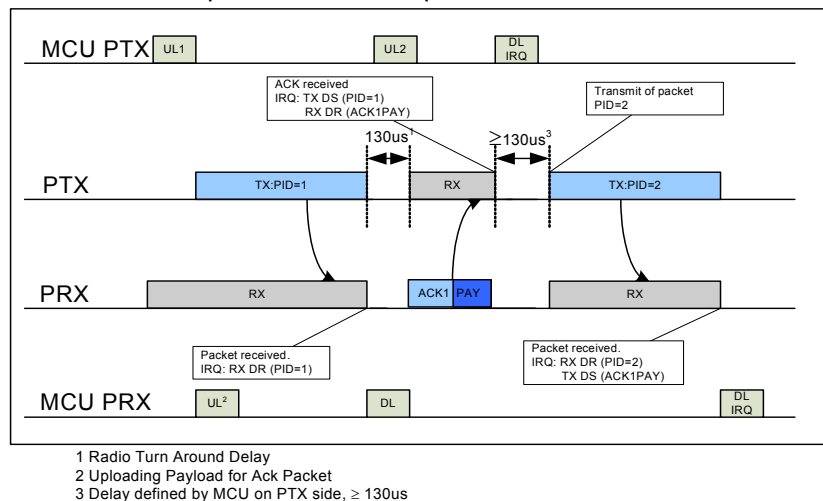


Figure 21. TX/RX cycles with ACK Payload and the according interrupts

7.8.5 Single transaction with ACK payload packet and lost packet

Figure 22. is a scenario where the first packet is lost and a retransmission is needed before the `RX_DR` IRQ on the PRX side is asserted. For the PTX both the `TX_DS` and `RX_DR` IRQ are asserted after the ACK packet is received. After the second packet (PID=2) is received on the PRX side both the `RX_DR` (PID=2) and `TX_DS` (ACK packet payload) IRQ are asserted.

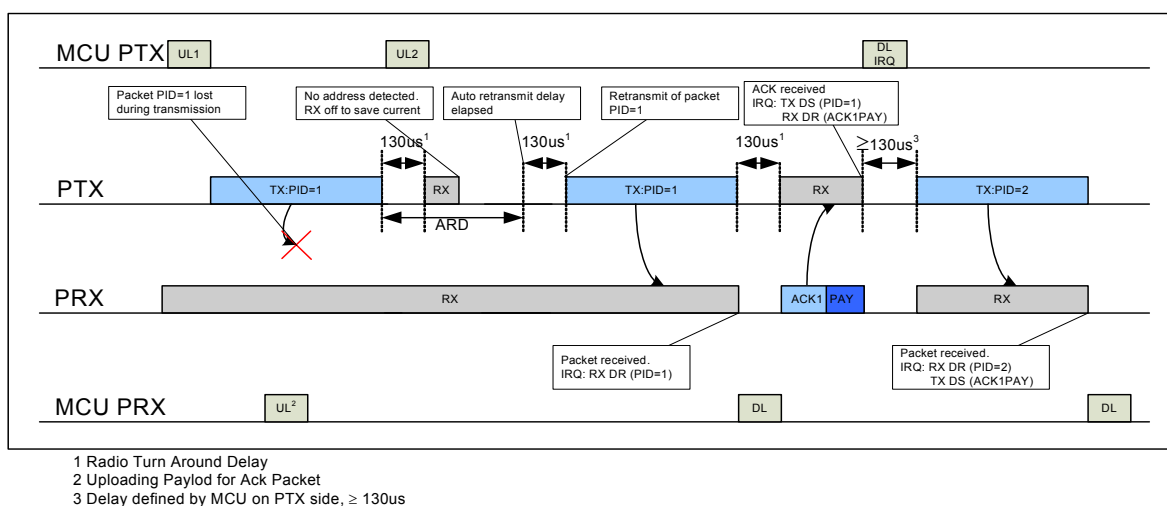


Figure 22. TX/RX cycles and the according interrupts when the packet transmission fails

7.8.6 Two transactions with ACK payload packet and the first ACK packet lost

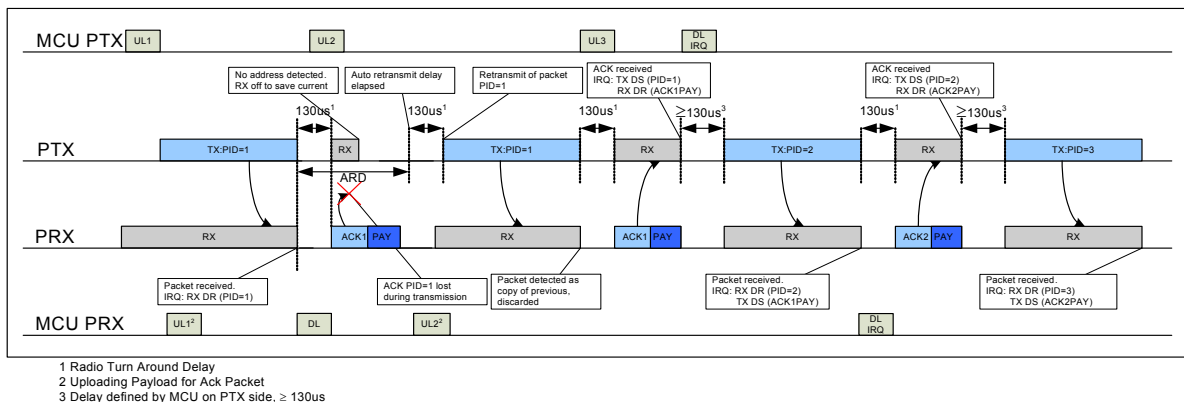


Figure 23. TX/RX cycles with ACK Payload and the according interrupts when the ACK packet fails

In Figure 23, the ACK packet is lost and a retransmission is needed before the `TX_DS` IRQ is asserted, but the `RX_DR` IRQ is asserted immediately. The retransmission of the packet (PID=1) results in a discarded packet. For the PTX both the `TX_DS` and `RX_DR` IRQ are asserted after the second transmission of ACK, which is received. After the second packet (PID=2) is received on the PRX both the `RX_DR` (PID=2) and `TX_DS` (ACK1PAY) IRQ is asserted. The callouts explain the different events and interrupts.

7.8.7 Two transactions where max retransmissions is reached

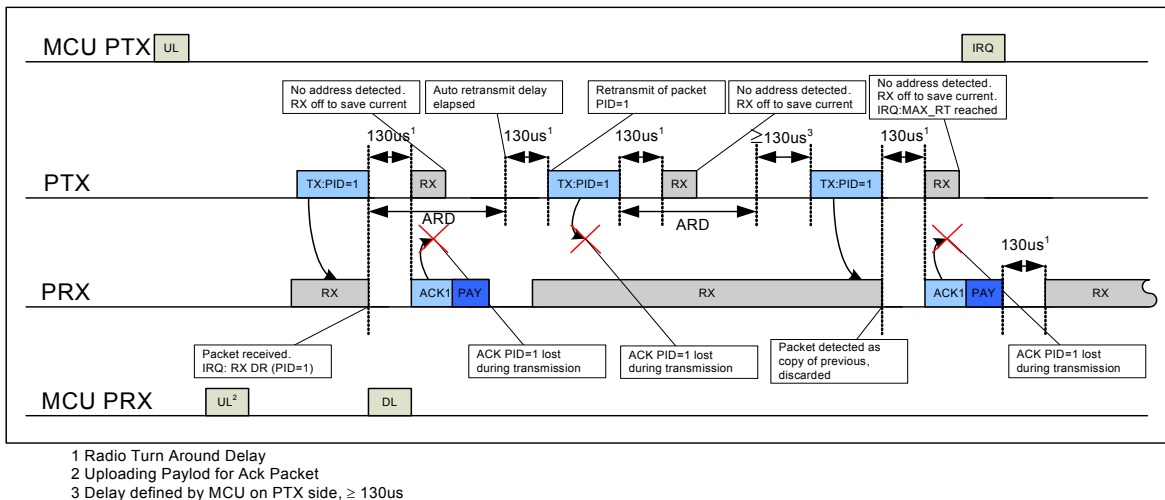


Figure 24. TX/RX cycles with ACK Payload and the according interrupts when the transmission fails. ARC is set to 2.

`MAX_RT` IRQ is asserted if the auto retransmit counter (`ARC_CNT`) exceeds the programmed maximum limit (`ARC`). In Figure 24, the packet transmission ends with a `MAX_RT` IRQ. The payload in TX FIFO is NOT removed and the MCU decides the next step in the protocol. A toggle of the `CE` starts a new transmitting sequence of the same packet. The payload can be removed from the TX FIFO using the `FLUSH_TX` command.

7.9 Compatibility with ShockBurst™

You must disable Enhanced ShockBurst™ for backward compatibility with the nRF2401A, nRF2402, nRF24E1 and nRF24E2. Set the register `EN_AA` = 0x00 and `ARC` = 0 to disable Enhanced ShockBurst™. In addition, the nRF24L01+ air data rate must be set to 1Mbps or 250kbps.

7.9.1 ShockBurst™ packet format

[Figure 25](#) shows the packet format with MSB to the left.



Figure 25. A ShockBurst™ packet compatible with nRF2401/nRF2402/nRF24E1/nRF24E2 devices.

The ShockBurst™ packet format has a preamble, address, payload and CRC field that are the same as the Enhanced ShockBurst™ packet format described in [section 7.3 on page 28](#).

The differences between the ShockBurst™ packet and the Enhanced ShockBurst™ packet are:

- The 9 bit Packet Control Field is not present in the ShockBurst™ packet format.
- The CRC is optional in the ShockBurst™ packet format and is controlled by the `EN_CRC` bit in the `CONFIG` register.

8 Data and Control Interface

The data and control interface gives you access to all the features in the nRF24L01+. The data and control interface consists of the following six 5V tolerant digital signals:

- **IRQ** (this signal is active low and controlled by three maskable interrupt sources)
- **CE** (this signal is active high and used to activate the chip in RX or TX mode)
- **CSN** (SPI signal)
- **SCK** (SPI signal)
- **MOSI** (SPI signal)
- **MISO** (SPI signal)

Using 1 byte SPI commands, you can activate the nRF24L01+ data FIFOs or the register map during all modes of operation.

8.1 Features

- Special SPI commands for quick access to the most frequently used features
- 0-10Mbps 4-wire SPI
- 8 bit command set
- Easily configurable register map
- Full three level FIFO for both TX and RX direction

8.2 Functional description

The SPI is a standard SPI with a maximum data rate of 10Mbps.

8.3 SPI operation

This section describes the SPI commands and timing.

8.3.1 SPI commands

The SPI commands are shown in [Table 20](#). Every new command must be started by a high to low transition on **CSN**.

The **STATUS** register is serially shifted out on the **MISO** pin simultaneously to the SPI command word shifting to the **MOSI** pin.

The serial shifting SPI commands is in the following format:

<**Command word**: MSBit to LSBit (one byte)>

<**Data bytes**: LSByte to MSByte, MSBit in each byte first>

See [Figure 26. on page 52](#) and [Figure 27. on page 52](#) for timing information.

Command name	Command word (binary)	# Data bytes	Operation
R_REGISTER	000A AAAA	1 to 5 LSByte first	Read command and <i>status</i> registers. AAAAA = 5 bit Register Map Address
W_REGISTER	001A AAAA	1 to 5 LSByte first	Write command and status registers. AAAAA = 5 bit Register Map Address Executable in power down or standby modes only.
R_RX_PAYLOAD	0110 0001	1 to 32 LSByte first	Read RX-payload: 1 – 32 bytes. A read operation always starts at byte 0. Payload is deleted from FIFO after it is read. Used in RX mode.
W_TX_PAYLOAD	1010 0000	1 to 32 LSByte first	Write TX-payload: 1 – 32 bytes. A write operation always starts at byte 0 used in TX payload.
FLUSH_TX	1110 0001	0	Flush TX FIFO, used in TX mode
FLUSH_RX	1110 0010	0	Flush RX FIFO, used in RX mode Should not be executed during transmission of acknowledge, that is, acknowledge package will not be completed.
REUSE_TX_PL	1110 0011	0	Used for a PTX device Reuse last transmitted payload. TX payload reuse is active until W_TX_PAYLOAD or FLUSH TX is executed. TX payload reuse must not be activated or deactivated during package transmission.
R_RX_PL_WID ^a	0110 0000	1	Read RX payload width for the top R_RX_PAYLOAD in the RX FIFO. Note: Flush RX FIFO if the read value is larger than 32 bytes.
W_ACK_PAYLOAD ^a	1010 1PPP	1 to 32 LSByte first	Used in RX mode. Write Payload to be transmitted together with ACK packet on PIPE PPP. (PPP valid in the range from 000 to 101). Maximum three ACK packet payloads can be pending. Payloads with same PPP are handled using first in - first out principle. Write payload: 1– 32 bytes. A write operation always starts at byte 0.
W_TX_PAYLOAD_NOACK ^a	1011 0000	1 to 32 LSByte first	Used in TX mode. Disables AUTOACK on this specific packet.
NOP	1111 1111	0	No Operation. Might be used to read the STATUS register

a. The bits in the FEATURE register shown in [Table 28. on page 63](#) have to be set.

Table 20. Command set for the nRF24L01+ SPI

The W_REGISTER and R_REGISTER commands operate on single or multi-byte registers. When accessing multi-byte registers read or write to the MSBit of LSByte first. You can terminate the writing before all bytes in a multi-byte register are written, leaving the unwritten MSByte(s) unchanged. For example, the LSByte of RX_ADDR_P0 can be modified by writing only one byte to the RX_ADDR_P0 register. The content of the *status* register is always read to *MISO* after a high to low transition on *CSN*.

Note: The 3 bit pipe information in the `STATUS` register is updated during the `IRQ` pin high to low transition. The pipe information is unreliable if the `STATUS` register is read during an `IRQ` pin high to low transition.

8.3.2 SPI timing

SPI operation and timing is shown in [Figure 26.](#) to [Figure 28.](#) and in [Table 22.](#) to [Table 27.](#) nRF24L01+ must be in a standby or power down mode before writing to the configuration registers.

In [Figure 26.](#) to [Figure 28.](#) the following abbreviations are used:

Abbreviation	Description
Cn	SPI command bit
Sn	<code>STATUS</code> register bit
Dn	Data Bit (Note: LSByte to MSByte, MSBit in each byte first)

Table 21. Abbreviations used in Figure 26. to Figure 28.



Figure 26. SPI read operation

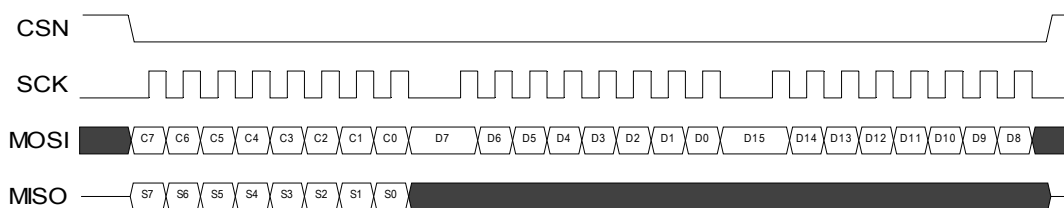


Figure 27. SPI write operation

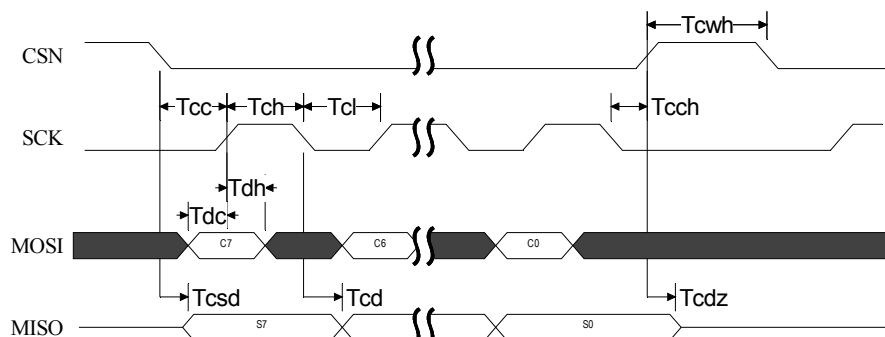


Figure 28. SPI NOP timing diagram

Figure 29. shows the R_{pull} and C_{load} that are referenced in Table 22. to Table 27.

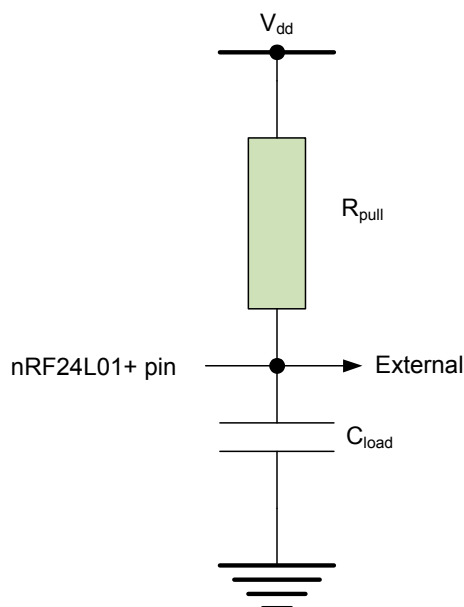


Figure 29. R_{pull} and C_{load}

Symbol	Parameters	Min.	Max	Units
Tdc	Data to sck Setup	2		ns
Tdh	sck to Data Hold	2		ns
Tcsd	csn to Data Valid		38	ns
Tcd	sck to Data Valid		55	ns
Tcl	sck Low Time	40		ns
Tch	sck High Time	40		ns
Fsck	sck Frequency	0	10	MHz
Tr,Tf	sck Rise and Fall		100	ns
Tcc	csn to sck Setup	2		ns
Tcch	sck to csn Hold	2		ns
Tcwh	csn Inactive time	50		ns
Tcdz	csn to Output High Z		38	ns

Table 22. SPI timing parameters ($C_{load} = 5pF$)

Symbol	Parameters	Min.	Max	Units
Tdc	Data to sck Setup	2		ns
Tdh	sck to Data Hold	2		ns
Tcsd	csn to Data Valid		42	ns
Tcd	sck to Data Valid		58	ns
Tcl	sck Low Time	40		ns
Tch	sck High Time	40		ns
Fsck	sck Frequency	0	8	MHz
Tr,Tf	sck Rise and Fall		100	ns
Tcc	csn to sck Setup	2		ns

Symbol	Parameters	Min.	Max	Units
Tcch	sck to csN Hold	2		ns
Tcwh	csN Inactive time	50		ns
Tcdz	csN to Output High Z		42	ns

Table 23. SPI timing parameters ($C_{load} = 10pF$)

Symbol	Parameters	Min.	Max	Units
Tdc	Data to sck Setup	2		ns
Tdh	sck to Data Hold	2		ns
Tcsd	csN to Data Valid		75	ns
Tcd	sck to Data Valid		86	ns
Tcl	sck Low Time	40		ns
Tch	sck High Time	40		ns
Fsck	sck Frequency	0	5	MHz
Tr,Tf	sck Rise and Fall		100	ns
Tcc	csN to sck Setup	2		ns
Tcch	sck to csN Hold	2		ns
Tcwh	csN Inactive time	50		ns
Tcdz	csN to Output High Z		75	ns

Table 24. SPI timing parameters ($R_{pull} = 10k\Omega$, $C_{load} = 50pF$)

Symbol	Parameters	Min.	Max	Units
Tdc	Data to sck Setup	2		ns
Tdh	sck to Data Hold	2		ns
Tcsd	csN to Data Valid		116	ns
Tcd	sck to Data Valid		123	ns
Tcl	sck Low Time	40		ns
Tch	sck High Time	40		ns
Fsck	sck Frequency	0	4	MHz
Tr,Tf	sck Rise and Fall		100	ns
Tcc	csN to sck Setup	2		ns
Tcch	sck to csN Hold	2		ns
Tcwh	csN Inactive time	50		ns
Tcdz	csN to Output High Z		116	ns

Table 25. SPI timing parameters ($R_{pull} = 10k\Omega$, $C_{load} = 100pF$)

Symbol	Parameters	Min.	Max	Units
Tdc	Data to sck Setup	2		ns
Tdh	sck to Data Hold	2		ns
Tcsd	csn to Data Valid		75	ns
Tcd	sck to Data Valid		85	ns
Tcl	sck Low Time	40		ns
Tch	sck High Time	40		ns
Fsck	sck Frequency	0	5	MHz
Tr,Tf	sck Rise and Fall		100	ns
Tcc	csn to sck Setup	2		ns
Tcch	sck to csn Hold	2		ns
Tcwh	csn Inactive time	50		ns
Tcdz	csn to Output High Z		75	ns

Table 26. SPI timing parameters ($R_{pull} = 50k\Omega$, $C_{load} = 50pF$)

Symbol	Parameters	Min.	Max	Units
Tdc	Data to sck Setup	2		ns
Tdh	sck to Data Hold	2		ns
Tcsd	csn to Data Valid		116	ns
Tcd	sck to Data Valid		121	ns
Tcl	sck Low Time	40		ns
Tch	sck High Time	40		ns
Fsck	sck Frequency	0	4	MHz
Tr,Tf	sck Rise and Fall		100	ns
Tcc	csn to sck Setup	2		ns
Tcch	sck to csn Hold	2		ns
Tcwh	csn Inactive time	50		ns
Tcdz	csn to Output High Z		116	ns

Table 27. SPI timing parameters ($R_{pull} = 50k\Omega$, $C_{load} = 100pF$)

8.4 Data FIFO

The data FIFOs store transmitted payloads (TX FIFO) or received payloads that are ready to be clocked out (RX FIFO). The FIFOs are accessible in both PTX mode and PRX mode.

The following FIFOs are present in nRF24L01+:

- TX three level, 32 byte FIFO
- RX three level, 32 byte FIFO

Both FIFOs have a controller and are accessible through the SPI by using dedicated SPI commands. A TX FIFO in PRX can store payloads for ACK packets to three different PTX devices. If the TX FIFO contains more than one payload to a pipe, payloads are handled using the first in - first out principle. The TX FIFO in a PRX is blocked if all pending payloads are addressed to pipes where the link to the PTX is lost. In this case, the MCU can flush the TX FIFO using the `FLUSH_TX` command.

The RX FIFO in PRX can contain payloads from up to three different PTX devices and a TX FIFO in PTX can have up to three payloads stored.

You can write to the TX FIFO using these three commands; `W_TX_PAYLOAD` and `W_TX_PAYLOAD_NO_ACK` in PTX mode and `W_ACK_PAYLOAD` in PRX mode. All three commands provide access to the `TX_PLD` register (see [Table 28. on page 63](#). for details of this register).

The RX FIFO can be read by the command `R_RX_PAYLOAD` in PTX and PRX mode. This command provides access to the `RX_PLD` register.

The payload in TX FIFO in a PTX is not removed if the `MAX_RT` IRQ is asserted.

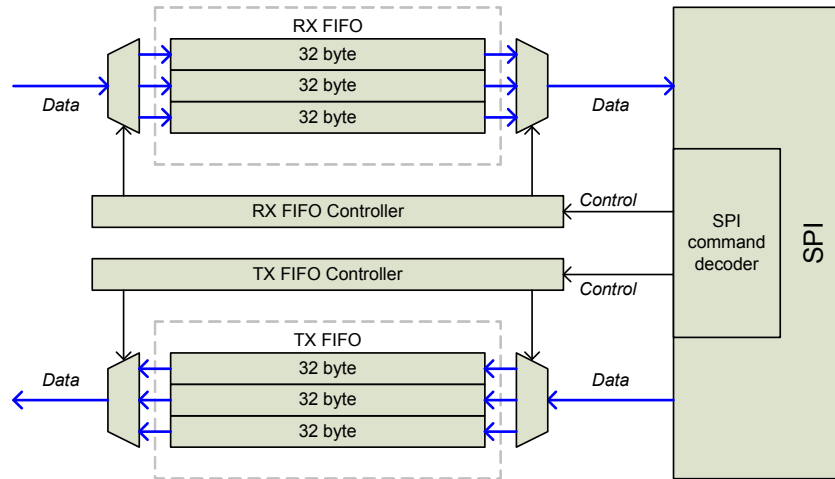


Figure 30. FIFO (RX and TX) block diagram

You can read if the TX and RX FIFO are full or empty in the `FIFO_STATUS` register.

8.5 Interrupt

The nRF24L01+ has an active low interrupt (`IRQ`) pin. The `IRQ` pin is activated when `TX_DS` IRQ, `RX_DR` IRQ or `MAX_RT` IRQ are set high by the state machine in the `STATUS` register. The `IRQ` pin resets when MCU writes '1' to the IRQ source bit in the `STATUS` register. The IRQ mask in the `CONFIG` register is used to select the IRQ sources that are allowed to assert the `IRQ` pin. By setting one of the MASK bits high, the corresponding IRQ source is disabled. By default all IRQ sources are enabled.

Note: The 3 bit pipe information in the `STATUS` register is updated during the `IRQ` pin high to low transition. The pipe information is unreliable if the `STATUS` register is read during an `IRQ` pin high to low transition.

9 Register Map

You can configure and control the radio by accessing the register map through the SPI.

9.1 Register map table

All undefined bits in the table below are redundant. They are read out as '0'.

Note: Addresses 18 to 1B are reserved for test purposes, altering them makes the chip malfunction.

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
00	CONFIG				Configuration Register
	Reserved	7	0	R/W	Only '0' allowed
	MASK_RX_DR	6	0	R/W	Mask interrupt caused by RX_DR 1: Interrupt not reflected on the IRQ pin 0: Reflect RX_DR as active low interrupt on the IRQ pin
	MASK_TX_DS	5	0	R/W	Mask interrupt caused by TX_DS 1: Interrupt not reflected on the IRQ pin 0: Reflect TX_DS as active low interrupt on the IRQ pin
	MASK_MAX_RT	4	0	R/W	Mask interrupt caused by MAX_RT 1: Interrupt not reflected on the IRQ pin 0: Reflect MAX_RT as active low interrupt on the IRQ pin
	EN_CRC	3	1	R/W	Enable CRC. Forced high if one of the bits in the EN_AA is high
	CRCO	2	0	R/W	CRC encoding scheme '0' - 1 byte '1' - 2 bytes
	PWR_UP	1	0	R/W	1: POWER UP, 0: POWER DOWN
	PRIM_RX	0	0	R/W	RX/TX control 1: PRX, 0: PTX
01	EN_AA Enhanced ShockBurst™				Enable 'Auto Acknowledgment' Function Disable this functionality to be compatible with nRF2401, see page 75
	Reserved	7:6	00	R/W	Only '00' allowed
	ENAA_P5	5	1	R/W	Enable auto acknowledgement data pipe 5
	ENAA_P4	4	1	R/W	Enable auto acknowledgement data pipe 4
	ENAA_P3	3	1	R/W	Enable auto acknowledgement data pipe 3
	ENAA_P2	2	1	R/W	Enable auto acknowledgement data pipe 2
	ENAA_P1	1	1	R/W	Enable auto acknowledgement data pipe 1
	ENAA_P0	0	1	R/W	Enable auto acknowledgement data pipe 0
02	EN_RXADDR				Enabled RX Addresses
	Reserved	7:6	00	R/W	Only '00' allowed
	ERX_P5	5	0	R/W	Enable data pipe 5.
	ERX_P4	4	0	R/W	Enable data pipe 4.
	ERX_P3	3	0	R/W	Enable data pipe 3.
	ERX_P2	2	0	R/W	Enable data pipe 2.

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
	ERX_P1	1	1	R/W	Enable data pipe 1.
	ERX_P0	0	1	R/W	Enable data pipe 0.
03	SETUP_AW				Setup of Address Widths (common for all data pipes)
	Reserved	7:2	000000	R/W	Only '000000' allowed
	AW	1:0	11	R/W	RX/TX Address field width '00' - Illegal '01' - 3 bytes '10' - 4 bytes '11' - 5 bytes LSByte is used if address width is below 5 bytes
04	SETUP_RETR				Setup of Automatic Retransmission
	ARD ^a	7:4	0000	R/W	Auto Retransmit Delay '0000' – Wait 250µS '0001' – Wait 500µS '0010' – Wait 750µS '1111' – Wait 4000µS (Delay defined from end of transmission to start of next transmission) ^b
	ARC	3:0	0011	R/W	Auto Retransmit Count '0000' – Re-Transmit disabled '0001' – Up to 1 Re-Transmit on fail of AA '1111' – Up to 15 Re-Transmit on fail of AA
05	RF_CH				RF Channel
	Reserved	7	0	R/W	Only '0' allowed
	RF_CH	6:0	0000010	R/W	Sets the frequency channel nRF24L01+ operates on
06	RF_SETUP				RF Setup Register
	CONT_WAVE	7	0	R/W	Enables continuous carrier transmit when high.
	Reserved	6	0	R/W	Only '0' allowed
	RF_DR_LOW	5	0	R/W	Set RF Data Rate to 250kbps. See RF_DR_HIGH for encoding.
	PLL_LOCK	4	0	R/W	Force PLL lock signal. Only used in test
	RF_DR_HIGH	3	1	R/W	Select between the high speed data rates. This bit is don't care if RF_DR_LOW is set. Encoding: [RF_DR_LOW, RF_DR_HIGH]: '00' – 1Mbps '01' – 2Mbps '10' – 250kbps '11' – Reserved

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
	RF_PWR	2:1	11	R/W	Set RF output power in TX mode '00' – -18dBm '01' – -12dBm '10' – -6dBm '11' – 0dBm
	Obsolete	0			Don't care
07	STATUS				Status Register (In parallel to the SPI command word applied on the MOSI pin, the STATUS register is shifted serially out on the MISO pin)
	Reserved	7	0	R/W	Only '0' allowed
	RX_DR	6	0	R/W	Data Ready RX FIFO interrupt. Asserted when new data arrives RX FIFO ^c . Write 1 to clear bit.
	TX_DS	5	0	R/W	Data Sent TX FIFO interrupt. Asserted when packet transmitted on TX. If AUTO_ACK is activated, this bit is set high only when ACK is received. Write 1 to clear bit.
	MAX_RT	4	0	R/W	Maximum number of TX retransmits interrupt Write 1 to clear bit. If MAX_RT is asserted it must be cleared to enable further communication.
	RX_P_NO	3:1	111	R	Data pipe number for the payload available for reading from RX_FIFO 000-101: Data Pipe Number 110: Not Used 111: RX FIFO Empty
	TX_FULL	0	0	R	TX FIFO full flag. 1: TX FIFO full. 0: Available locations in TX FIFO.
08	OBSERVE_TX				Transmit observe register
	PLOS_CNT	7:4	0	R	Count lost packets. The counter is overflow protected to 15, and discontinues at max until reset. The counter is reset by writing to RF_CH. See page 75 .
	ARC_CNT	3:0	0	R	Count retransmitted packets. The counter is reset when transmission of a new packet starts. See page 75 .
09	RPD				
	Reserved	7:1	000000	R	
	RPD	0	0	R	Received Power Detector. This register is called CD (Carrier Detect) in the nRF24L01. The name is different in nRF24L01+ due to the different input power level threshold for this bit. See section 6.4 on page 25 .
0A	RX_ADDR_P0	39:0	0xE7E7E7E7E7	R/W	Receive address data pipe 0. 5 Bytes maximum length. (LSByte is written first. Write the number of bytes defined by SETUP_AW)

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
0B	RX_ADDR_P1	39:0	0xC2C2C2C2	R/W	Receive address data pipe 1. 5 Bytes maximum length. (LSByte is written first. Write the number of bytes defined by SETUP_AW)
0C	RX_ADDR_P2	7:0	0xC3	R/W	Receive address data pipe 2. Only LSB. MSBytes are equal to RX_ADDR_P1[39:8]
0D	RX_ADDR_P3	7:0	0xC4	R/W	Receive address data pipe 3. Only LSB. MSBytes are equal to RX_ADDR_P1[39:8]
0E	RX_ADDR_P4	7:0	0xC5	R/W	Receive address data pipe 4. Only LSB. MSBytes are equal to RX_ADDR_P1[39:8]
0F	RX_ADDR_P5	7:0	0xC6	R/W	Receive address data pipe 5. Only LSB. MSBytes are equal to RX_ADDR_P1[39:8]
10	TX_ADDR	39:0	0xE7E7E7E7	R/W	Transmit address. Used for a PTX device only. (LSByte is written first) Set RX_ADDR_P0 equal to this address to handle automatic acknowledge if this is a PTX device with Enhanced ShockBurst™ enabled. See page 75 .
11	RX_PW_P0				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P0	5:0	0	R/W	Number of bytes in RX payload in data pipe 0 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte ... 32 = 32 bytes
12	RX_PW_P1				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P1	5:0	0	R/W	Number of bytes in RX payload in data pipe 1 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte ... 32 = 32 bytes
13	RX_PW_P2				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P2	5:0	0	R/W	Number of bytes in RX payload in data pipe 2 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte ... 32 = 32 bytes
14	RX_PW_P3				
	Reserved	7:6	00	R/W	Only '00' allowed

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
	RX_PW_P3	5:0	0	R/W	Number of bytes in RX payload in data pipe 3 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte ... 32 = 32 bytes
15	RX_PW_P4				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P4	5:0	0	R/W	Number of bytes in RX payload in data pipe 4 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte ... 32 = 32 bytes
16	RX_PW_P5				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P5	5:0	0	R/W	Number of bytes in RX payload in data pipe 5 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte ... 32 = 32 bytes
17	FIFO_STATUS				FIFO Status Register
	Reserved	7	0	R/W	Only '0' allowed
	TX_REUSE	6	0	R	Used for a PTX device Pulse the <code>rfce</code> high for at least 10µs to Reuse last transmitted payload. TX payload reuse is active until <code>W_TX_PAYLOAD</code> or <code>FLUSH_TX</code> is executed. <code>TX_REUSE</code> is set by the SPI command <code>REUSE_TX_PL</code> , and is reset by the SPI commands <code>W_TX_PAYLOAD</code> or <code>FLUSH_TX</code>
	TX_FULL	5	0	R	TX FIFO full flag. 1: TX FIFO full. 0: Available locations in TX FIFO.
	TX_EMPTY	4	1	R	TX FIFO empty flag. 1: TX FIFO empty. 0: Data in TX FIFO.
	Reserved	3:2	00	R/W	Only '00' allowed
	RX_FULL	1	0	R	RX FIFO full flag. 1: RX FIFO full. 0: Available locations in RX FIFO.
	RX_EMPTY	0	1	R	RX FIFO empty flag. 1: RX FIFO empty. 0: Data in RX FIFO.

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
N/A	ACK_PLD	255:0	X	W	Written by separate SPI command ACK packet payload to data pipe number PPP given in SPI command. Used in RX mode only. Maximum three ACK packet payloads can be pending. Payloads with same PPP are handled first in first out.
N/A	TX_PLD	255:0	X	W	Written by separate SPI command TX data payload register 1 - 32 bytes. This register is implemented as a FIFO with three levels. Used in TX mode only.
N/A	RX_PLD	255:0	X	R	Read by separate SPI command. RX data payload register. 1 - 32 bytes. This register is implemented as a FIFO with three levels. All RX channels share the same FIFO.
1C	DYNPD				Enable dynamic payload length
	Reserved	7:6	0	R/W	Only '00' allowed
	DPL_P5	5	0	R/W	Enable dynamic payload length data pipe 5. (Requires EN_DPL and ENAA_P5)
	DPL_P4	4	0	R/W	Enable dynamic payload length data pipe 4. (Requires EN_DPL and ENAA_P4)
	DPL_P3	3	0	R/W	Enable dynamic payload length data pipe 3. (Requires EN_DPL and ENAA_P3)

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
	DPL_P2	2	0	R/W	Enable dynamic payload length data pipe 2. (Requires EN_DPL and ENAA_P2)
	DPL_P1	1	0	R/W	Enable dynamic payload length data pipe 1. (Requires EN_DPL and ENAA_P1)
	DPL_P0	0	0	R/W	Enable dynamic payload length data pipe 0. (Requires EN_DPL and ENAA_P0)
1D	FEATURE			R/W	Feature Register
	Reserved	7:3	0	R/W	Only '00000' allowed
	EN_DPL	2	0	R/W	Enables Dynamic Payload Length
	EN_ACK_PAY ^d	1	0	R/W	Enables Payload with ACK
	EN_DYN_ACK	0	0	R/W	Enables the W_TX_PAYLOAD_NOACK command

- Please take care when setting this parameter. If the ACK payload is more than 15 byte in 2Mbps mode the ARD must be 500µS or more, if the ACK payload is more than 5byte in 1Mbps mode the ARD must be 500µS or more. In 250kbps mode (even when the payload is not in ACK) the ARD must be 500µS or more. Please see section [7.4.2 on page 33](#) for more information.
- This is the time the PTX is waiting for an ACK packet before a retransmit is made. The PTX is in RX mode for 250µS (500µS in 250kbps mode) to wait for address match. If the address match is detected, it stays in RX mode to the end of the packet, unless ARD elapses. Then it goes to standby-II mode for the rest of the specified ARD. After the ARD it goes to TX mode and then retransmits the packet.
- The RX_DR IRQ is asserted by a new packet arrival event. The procedure for handling this interrupt should be: 1) read payload through SPI, 2) clear RX_DR IRQ, 3) read FIFO_STATUS to check if there are more payloads available in RX FIFO, 4) if there are more data in RX FIFO, repeat from step 1).
- If ACK packet payload is activated, ACK packets have dynamic payload lengths and the Dynamic Payload Length feature should be enabled for pipe 0 on the PTX and PRX. This is to ensure that they receive the ACK packets with payloads. If the ACK payload is more than 15 byte in 2Mbps mode the ARD must be 500µS or more, and if the ACK payload is more than 5 byte in 1Mbps mode the ARD must be 500µS or more. In 250kbps mode (even when the payload is not in ACK) the ARD must be 500µS or more.

Table 28. Register map of nRF24L01+

10 Peripheral RF Information

This chapter describes peripheral circuitry and PCB layout requirements that are important for achieving optimum RF performance from the nRF24L01+.

10.1 Antenna output

The **ANT1** and **ANT2** output pins provide a balanced RF output to the antenna. The pins must have a DC path to **VDD_PA**, either through a RF choke or through the center point in a balanced dipole antenna. A load of $15\Omega + j88\Omega$ is recommended for maximum output power (0dBm). Lower load impedance (for instance, 50Ω) can be obtained by fitting a simple matching network between the load and **ANT1** and **ANT2**. A recommended matching network for 50Ω load impedance is described in [chapter 11 on page 66](#).

10.2 Crystal oscillator

A crystal used with the nRF24L01+ must fulfil the specifications in [Table 11. on page 19](#).

To achieve a crystal oscillator solution with low power consumption and fast start up time use a crystal with a low load capacitance specification. A lower C_0 also gives lower current consumption and faster start up time, but can increase the cost of the crystal. Typically $C_0 = 1.5\text{pF}$ at a crystal specified for $C_{0\text{max}} = 7.0\text{pF}$.

The crystal load capacitance, C_L , is given by:

$$C_L = \frac{C_1' \cdot C_2'}{C_1' + C_2'}, \text{ where } C_1' = C_1 + C_{\text{PCB1}} + C_{\text{I1}} \text{ and } C_2' = C_2 + C_{\text{PCB2}} + C_{\text{I2}}$$

C_1 and C_2 are SMD capacitors, see the application schematics in [Figure 32. on page 66](#). C_{PCB1} and C_{PCB2} are the layout parasitic on the circuit board. C_{I1} and C_{I2} are the internal capacitance load of the **XC1** and **XC2** pins respectively; the value is typically 1pF for both these pins.

10.3 nRF24L01+ crystal sharing with an MCU

Follow the rules described in sections [10.3.1](#) and [10.3.2](#) when using an MCU to drive the crystal reference input **XC1** of the nRF24L01+ transceiver.

10.3.1 Crystal parameters

The MCU sets the requirement of load capacitance C_L when it is driving the nRF24L01+ clock input. A frequency accuracy of $\pm 60\text{ppm}$ is required to get a functional radio link. The nRF24L01+ loads the crystal by 1pF in addition to the PCB routing.

10.3.2 Input crystal amplitude and current consumption

The input signal should not have amplitudes exceeding any rail voltage. Exceeding rail voltage excites the ESD structure and consequently, the radio performance degrades below specification. You must use an external DC block if you are testing the nRF24L01+ with a reference source that has no DC offset (which is usual with a RF source).

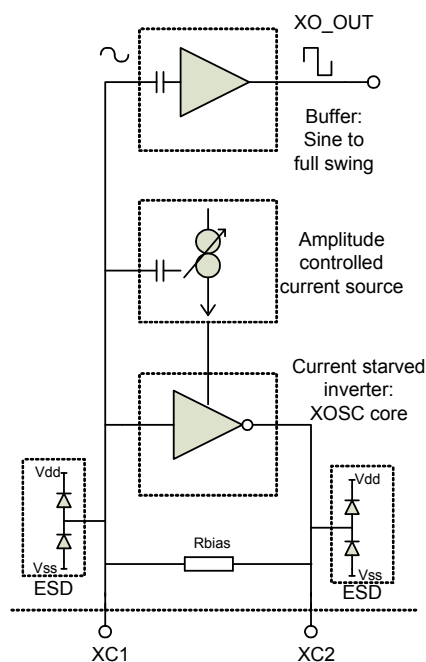


Figure 31. Principle of crystal oscillator

The nRF24L01+ crystal oscillator is amplitude regulated. It is recommended to use an input signal larger than 0.4V-peak to achieve low current consumption and good signal-to-noise ratio when using an external clock. **xc2** is not used and can be left as an open pin when clocked externally.

10.4 PCB layout and decoupling guidelines

A well designed PCB is necessary to achieve good RF performance. A poor layout can lead to loss of performance or functionality. You can download a fully qualified RF layout for the nRF24L01+ and its surrounding components, including matching networks, from www.nordicsemi.no.

A PCB with a minimum of two layers including a ground plane is recommended for optimum performance. The nRF24L01+ DC supply voltage should be decoupled as close as possible to the **VDD** pins with high performance RF capacitors, see [Table 29. on page 67](#). Mounting a large surface mount capacitor (for example, 4.7µF ceramic) in parallel with the smaller value capacitors is recommended. The nRF24L01+ supply voltage should be filtered and routed separately from the supply voltages of any digital circuitry.

Avoid long power supply lines on the PCB. All device grounds, **VDD** connections and **VDD** bypass capacitors must be connected as close as possible to the nRF24L01+ IC. The **VSS** pins should be connected directly to the ground plane for a PCB with a top-side RF ground plane. We recommend having via holes as close as possible to the **VSS** pads for a PCB with a bottom ground plane. A minimum of one via hole should be used for each **VSS** pin.

Full swing digital data or control signals should not be routed close to the crystal or the power supply lines. The exposed die attach pad is a ground pad connected to the IC substrate die ground and is intentionally not used in our layouts. We recommend to keep it unconnected.

nRF24L01+ with single ended matching network crystal, bias resistor, and decoupling capacitors.

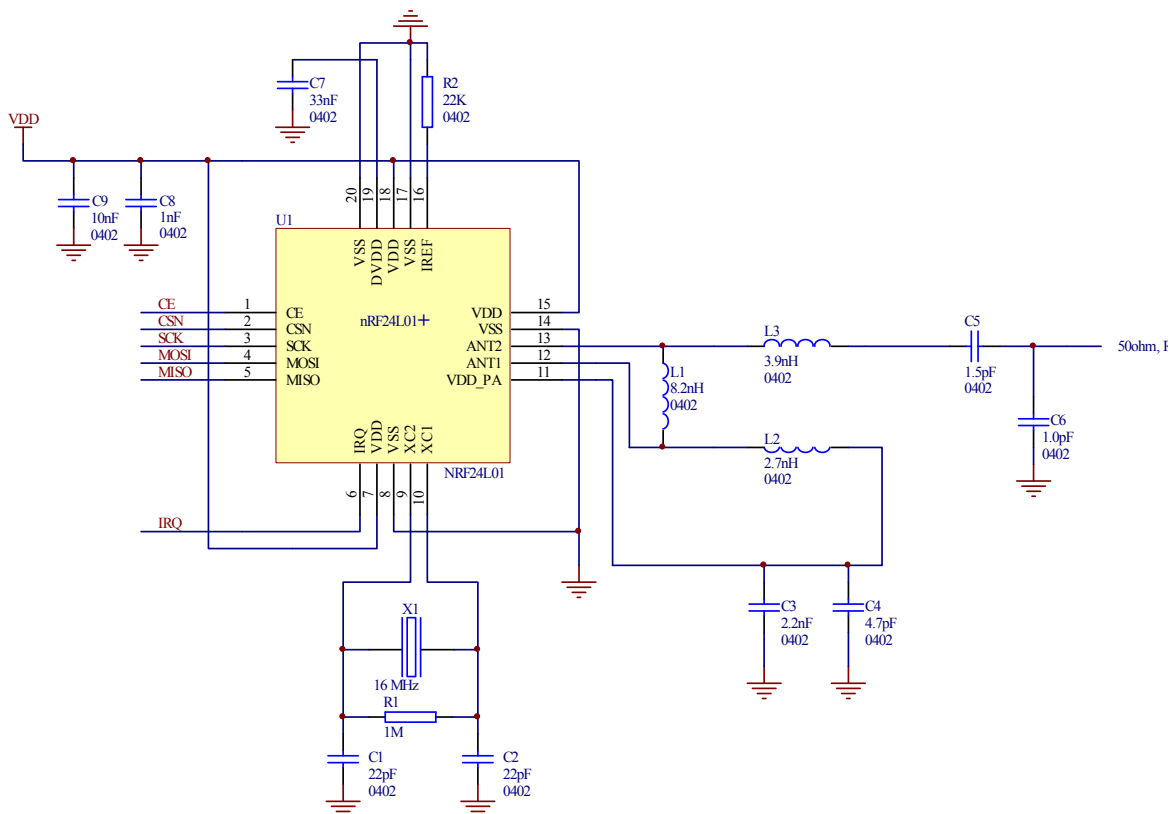


Figure 32. nRF24L01+ schematic for RF layouts with single ended 50Ω RF output

Part	Designator	Footprint	Description
22pF ^a	C1	0402	NPO, +/- 2%
22pF ^a	C2	0402	NPO, +/- 2%
2.2nF	C3	0402	X7R, +/- 10%
4.7pF	C4	0402	NPO, +/- 0.25pF
1.5pF	C5	0402	NPO, +/- 0.1pF
1.0pF	C6	0402	NPO, +/- 0.1pF
33nF	C7	0402	X7R, +/- 10%
1nF	C8	0402	X7R, +/- 10%
10nF	C9	0402	X7R, +/- 10%
8.2nH	L1	0402	chip inductor +/- 5%
2.7nH	L2	0402	chip inductor +/- 5%
3.9nH	L3	0402	chip inductor +/- 5%
Not mounted ^b	R1	0402	
22kΩ	R2	0402	+/-1%
nRF24L01+	U1	QFN20 4x4	
16MHz	X1		+/-60ppm, C _L =12pF

- a. C1 and C2 must have values that match the crystals load capacitance, C_L.
- b. The nRF24L01+ and nRF24L01 application example and BOM are the same with the exception of R1. R1 can be mounted for backward compatibility with nRF24L01. The use of a 1Mohm resistor externally does not have any impact on crystal performance.

Table 29. Recommended components (BOM) in nRF24L01+ with antenna matching network

11.1 PCB layout examples

[Figure 33.](#), [Figure 34.](#) and [Figure 35.](#) show a PCB layout example for the application schematic in [Figure 32.](#)

A double-sided FR-4 board of 1.6mm thickness is used. This PCB has a ground plane on the bottom layer. Additionally, there are ground areas on the component side of the board to ensure sufficient grounding of critical components. A large number of via holes connect the top layer ground areas to the bottom layer ground plane.

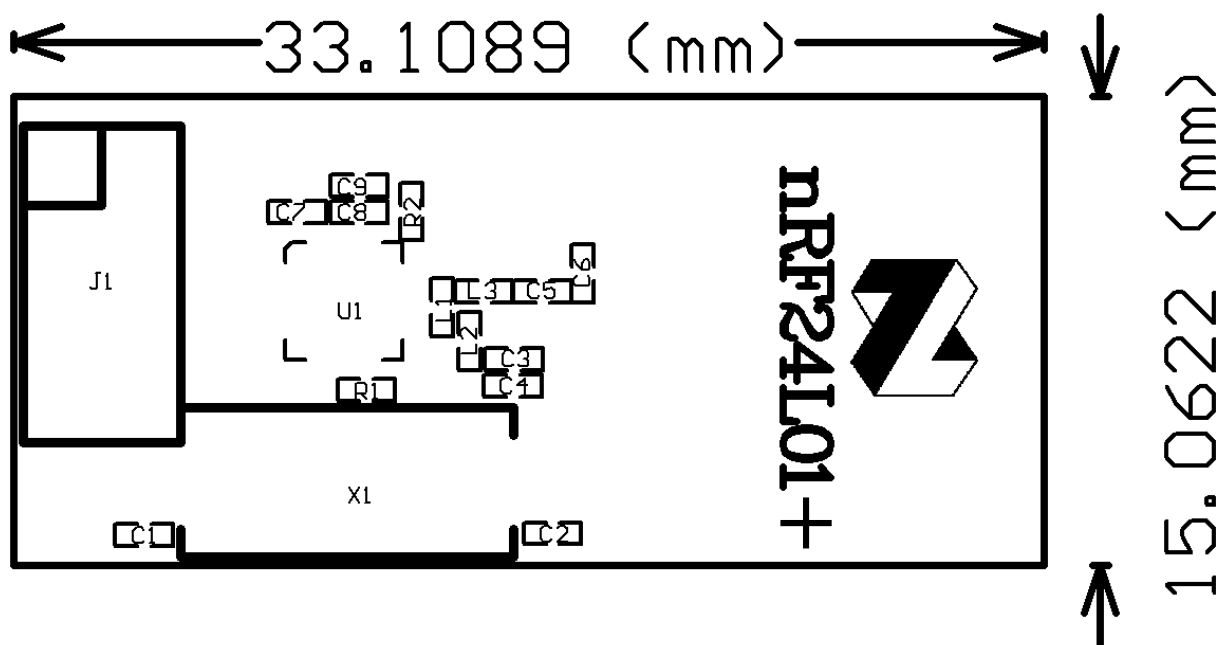


Figure 33. Top overlay (nRF24L01+ RF layout with single ended connection to PCB antenna and 0402 size passive components)

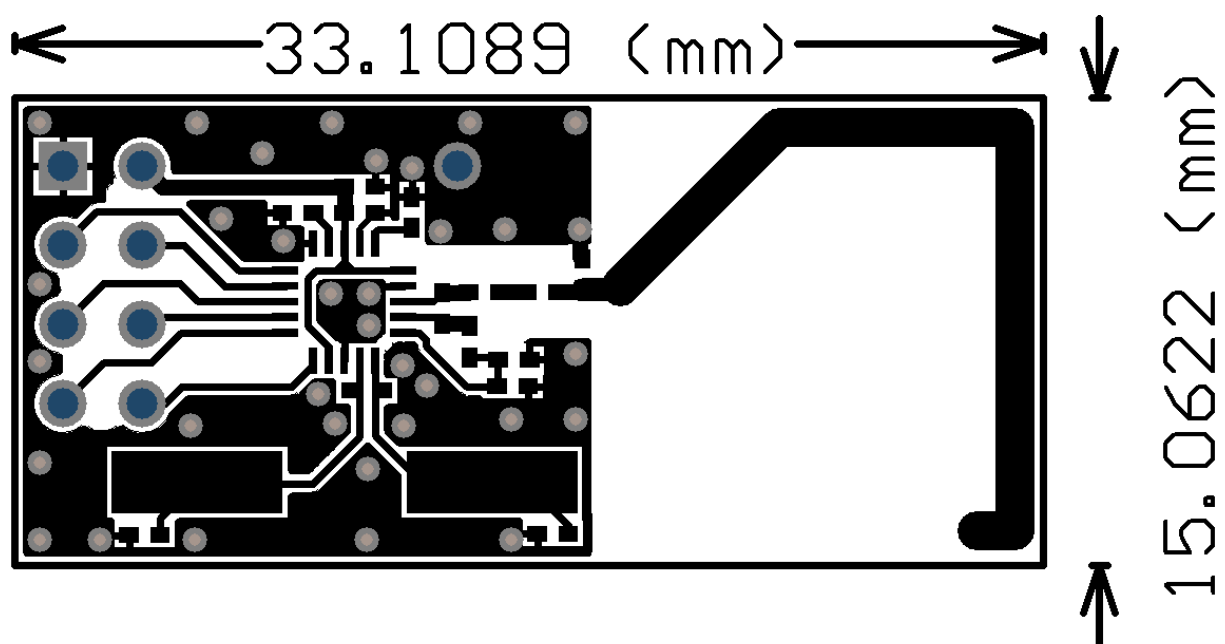


Figure 34. Top layer (nRF24L01+ RF layout with single ended connection to PCB antenna and 0402 size passive components)

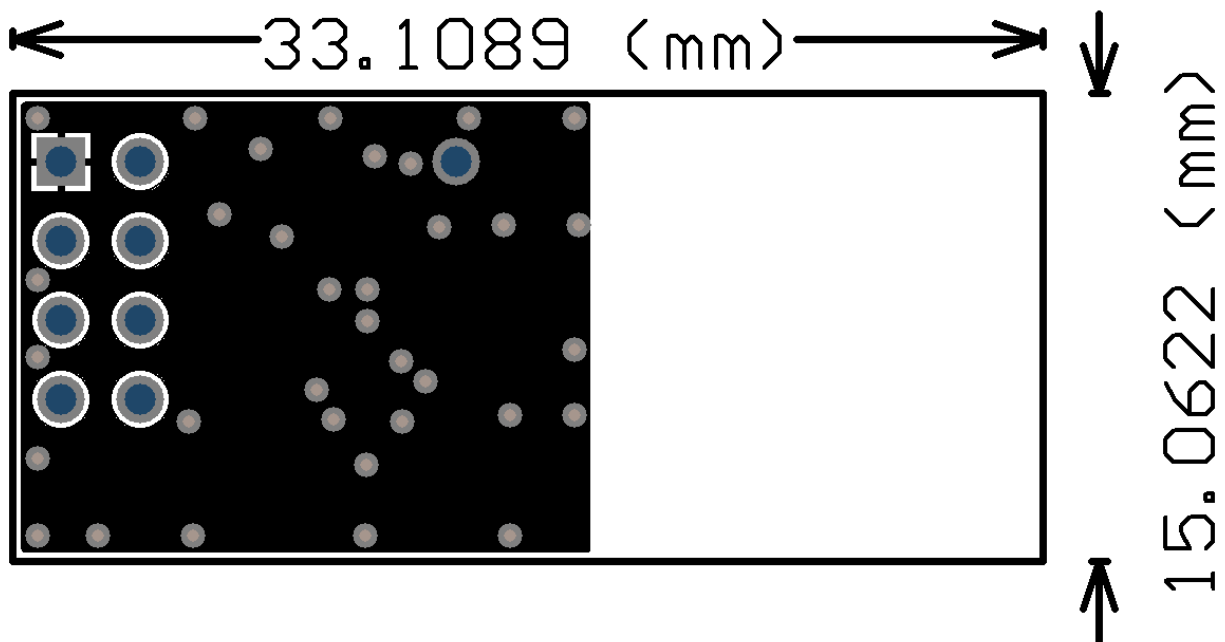


Figure 35. Bottom layer (nRF24L01+ RF layout with single ended connection to PCB antenna and 0402 size passive components)

The next figure ([Figure 36](#), [Figure 37](#), and [Figure 38](#).) is for the SMA output to have a board for direct measurements at a 50Ω SMA connector.

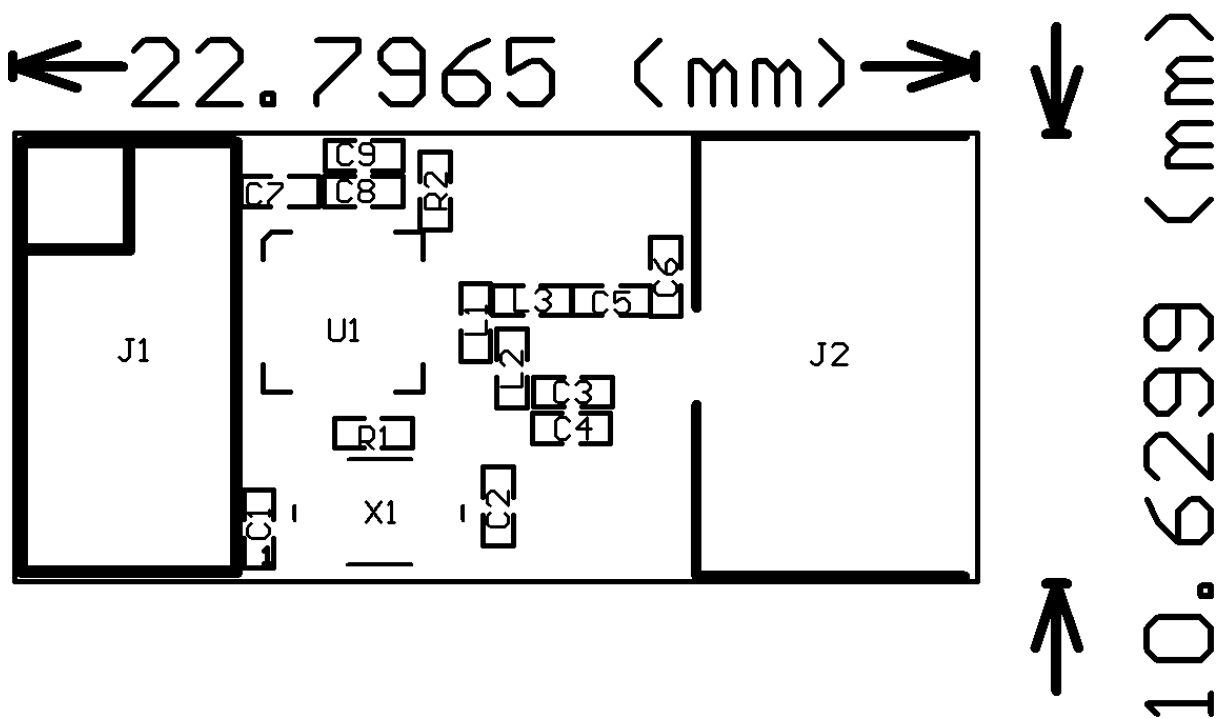


Figure 36. Top Overlay (Module with OFM crystal and SMA connector)

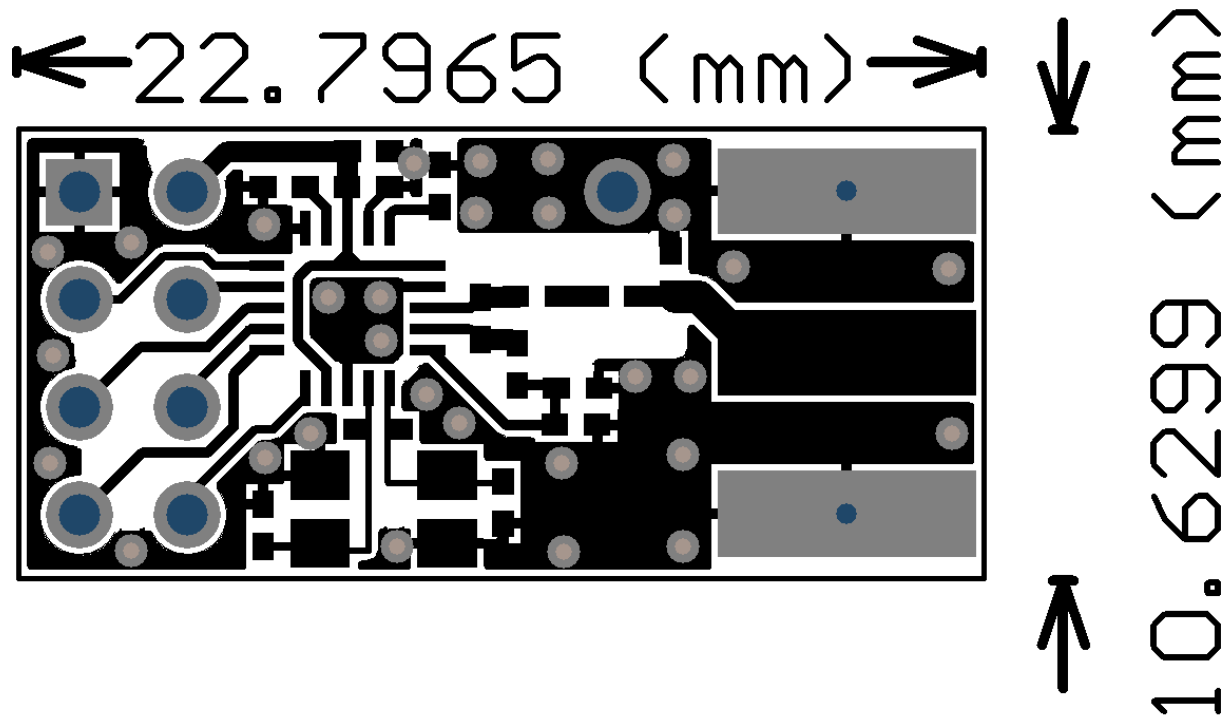


Figure 37. Top Layer (Module with OFM crystal and SMA connector)

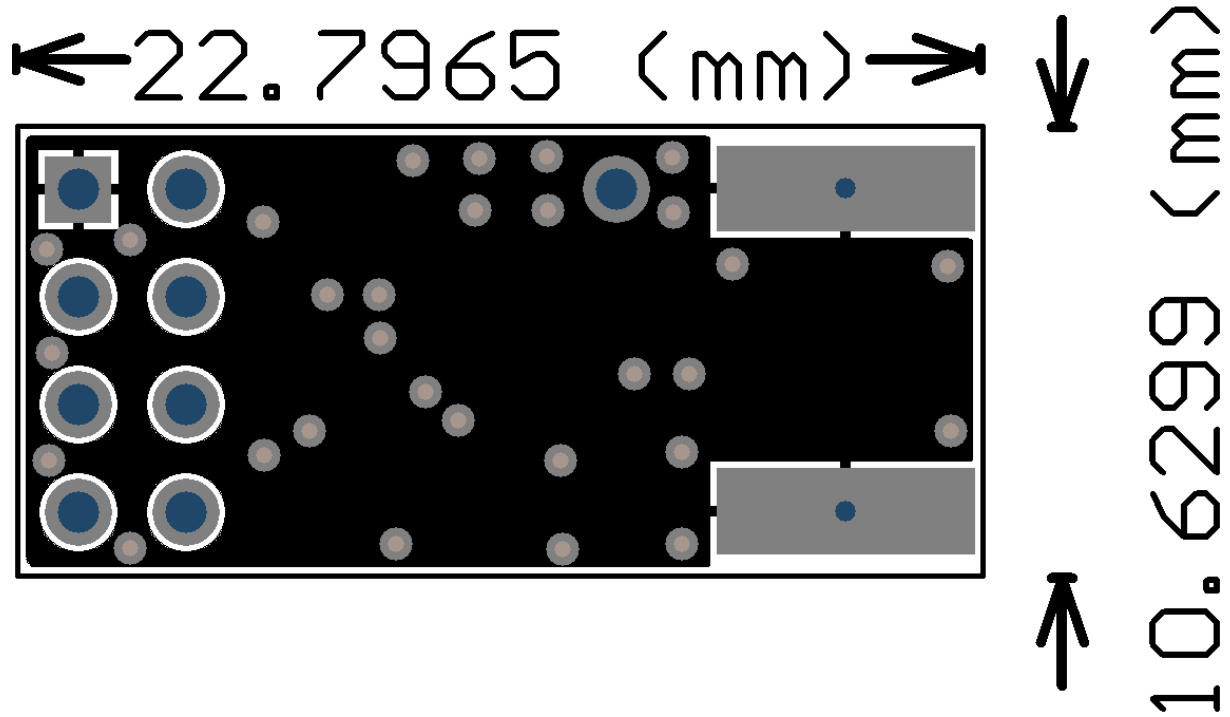
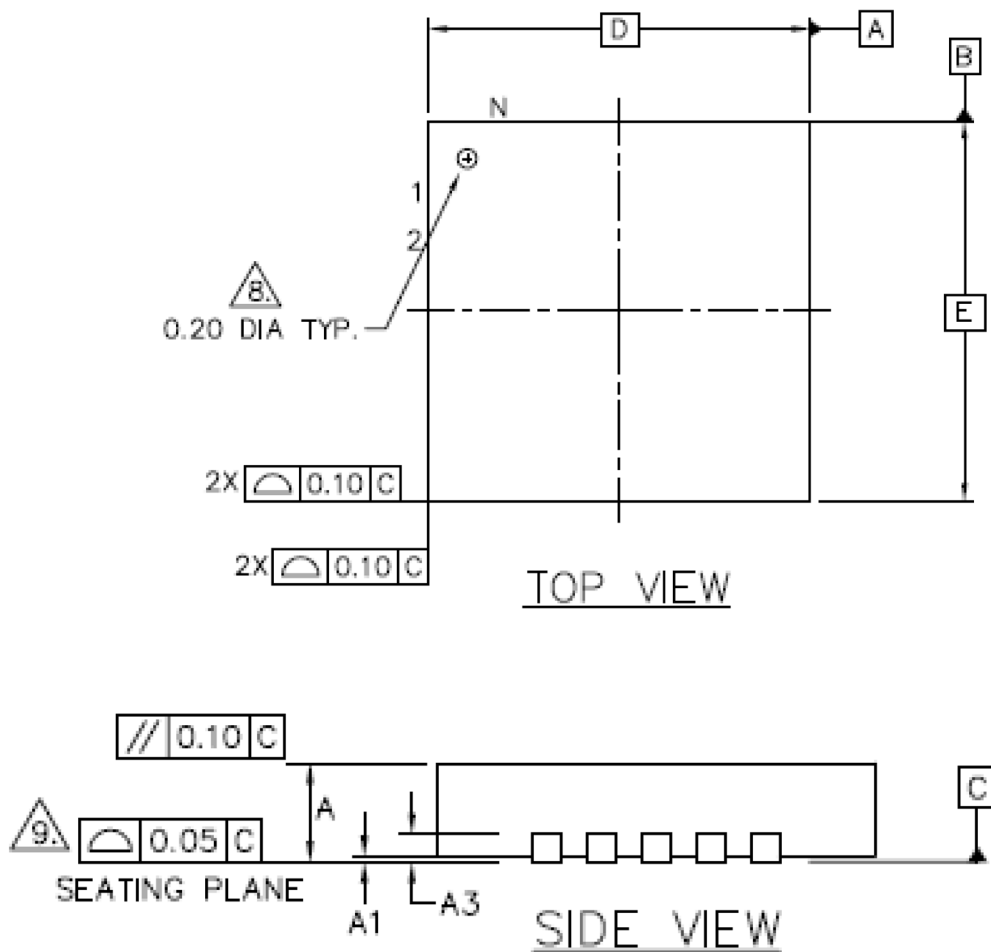
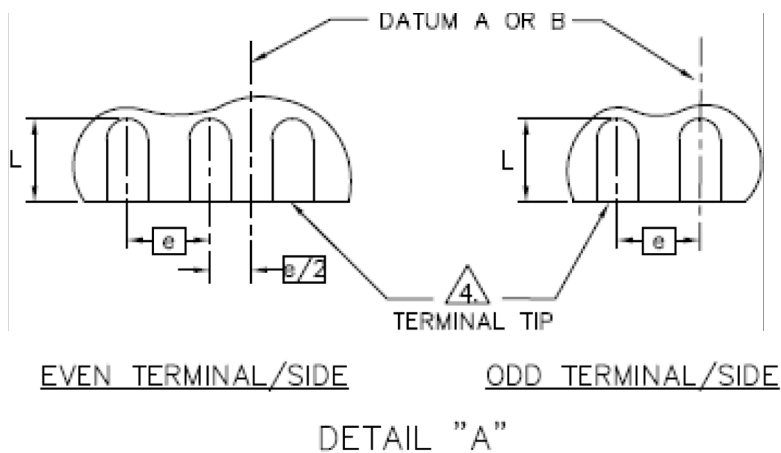
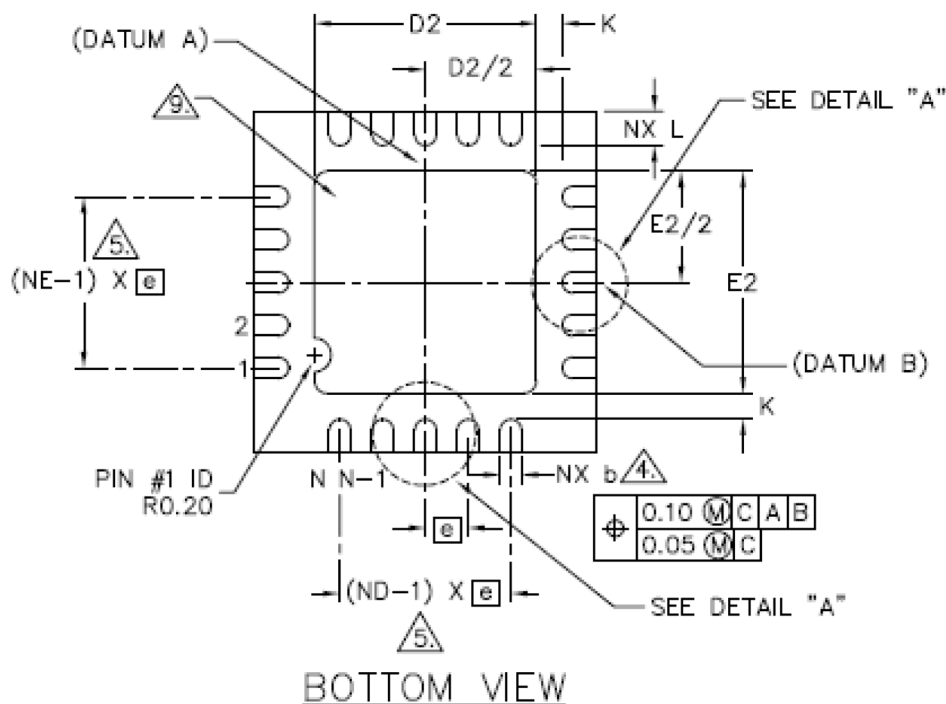


Figure 38. Bottom Layer (Module with OFM crystal and SMA connector)

12 Mechanical specifications

nRF24L01+ uses the QFN20 4x4 package, with matt tin plating.





Package Type		A	A1	A3	K	D/E	e	D2/E2	L	L1	b
Saw QFN20 (4x4 mm)	Min.	0.80	0.00					2.50	0.35		0.18
	Typ.	0.85	0.02	0.20	0.20	4.0	0.5 BSC	2.60	0.40	0.15	0.25
	Max	0.95	0.05	REF.	min.	BSC ^a		2.70	0.45	max	0.30

a. BSC: Basic Spacing between Centers, ref. JEDEC standard 95, page 4.17-11/A

Figure 39. nRF24L01+ Package Outline

13 Ordering information

13.1 Package marking

n	R	F		A	X
2	4	L	0	1	+
Y	Y	W	W	L	L

13.2 Abbreviations

Abbreviation	Definition
nRF	Fixed text
A	Variable Build Code, that is, unique code for production sites, package type and test platform
X	"X" grade, that is, Engineering Samples (optional)
YY	2 digit Year number
WW	2 digit Week number
LL	2 letter wafer lot number code

13.3 Product options

13.3.1 RF silicon

Ordering code	Package	Container	MOQ ^a
nRF24L01P-SAMPLE	4x4mm 20-pin QFN, lead free (green)	Sample box	5
nRF24L01P-T	4x4mm 20-pin QFN, lead free (green)	Tray	490
nRF24L01P-R	4x4mm 20-pin QFN, lead free (green)	Tape and reel	4000
nRF24L01P-R7	4x4mm 20-pin QFN, lead free (green)	Tape and reel	1500

a. Minimum Order Quantity

Table 30. nRF24L01+ RF silicon options

13.3.2 Development tools

Type Number	Description	Version
nRF24L01P-EVKIT	nRF24L01+ Evaluation kit	
nRF24L01P-UPGRADE	nRF24L01+ Upgrade kit for owners of nRF24L01-EVKIT	
nRF24L01P-MODULE-SMA	nRF24L01+ Evaluation kit module with SMA antenna	
nRF24L01P-MODULE-PCB	nRF24L01+ Evaluation kit module with PCB antenna	

Table 31. nRF24L01+ solution options

14 Glossary of Terms

Term	Description
ACK	Acknowledgement
ACS	Adjacent Channel Selectivity
AGC	Automatic Gain Control
ART	Auto Re-Transmit
CD	Carrier Detect
CE	Chip Enable
CLK	Clock
CRC	Cyclic Redundancy Check
CSN	Chip Select NOT
ESB	Enhanced ShockBurst™
GFSK	Gaussian Frequency Shift Keying
IM	Intermodulation
IRQ	Interrupt Request
ISM	Industrial-Scientific-Medical
LNA	Low Noise Amplifier
LSB	Least Significant Bit
LSByte	Least Significant Byte
Mbps	Megabit per second
MCU	Microcontroller Unit
MISO	Master In Slave Out
MOSI	Master Out Slave In
MSB	Most Significant Bit
MSByte	Most Significant Byte
PCB	Printed Circuit Board
PID	Packet Identity Bits
PLD	Payload
PRX	Primary RX
PTX	Primary TX
PWR_DWN	Power Down
PWR_UP	Power Up
RoHS	Restriction of use of Certain Hazardous Substances
RPD	Received Power Detector
RX	Receive
RX_DR	Receive Data Ready
SPI	Serial Peripheral Interface
TX	Transmit
TX_DS	Transmit Data Sent

Table 32. Glossary

Appendix A - Enhanced ShockBurst™ - Configuration and communication example

Enhanced ShockBurst™ transmitting payload

1. Set the configuration bit `PRIM_RX` low.
2. When the application MCU has data to transmit, clock the address for the receiving node (`TX_ADDR`) and payload data (`TX_PLD`) into nRF24L01+ through the SPI. The width of TX-payload is counted from the number of bytes written into the TX FIFO from the MCU. `TX_PLD` must be written continuously while holding `CSN` low. `TX_ADDR` does not have to be rewritten if it is unchanged from last transmit. If the PTX device shall receive acknowledge, configure data pipe 0 to receive the ACK packet. The RX address for data pipe 0 (`RX_ADDR_P0`) must be equal to the TX address (`TX_ADDR`) in the PTX device. For the example in [Figure 14. on page 41](#) perform the following address settings for the TX5 device and the RX device:
TX5 device: `TX_ADDR = 0xB3B4B5B605`
TX5 device: `RX_ADDR_P0 = 0xB3B4B5B605`
RX device: `RX_ADDR_P5 = 0xB3B4B5B605`
3. A high pulse on `CE` starts the transmission. The minimum pulse width on `CE` is 10µs.
4. nRF24L01+ ShockBurst™:
 - ▶ Radio is powered up.
 - ▶ 16MHz internal clock is started.
 - ▶ RF packet is completed (see the packet description).
 - ▶ Data is transmitted at high speed (1Mbps or 2Mbps configured by MCU).
5. If auto acknowledgement is activated (`ENAA_P0=1`) the radio goes into RX mode immediately, unless the `NO_ACK` bit is set in the received packet. If a valid packet is received in the valid acknowledgement time window, the transmission is considered a success. The `TX_DS` bit in the `STATUS` register is set high and the payload is removed from TX FIFO. If a valid ACK packet is not received in the specified time window, the payload is retransmitted (if auto retransmit is enabled). If the auto retransmit counter (`ARC_CNT`) exceeds the programmed maximum limit (`ARC`), the `MAX_RT` bit in the `STATUS` register is set high. The payload in TX FIFO is NOT removed. The `IRQ` pin is active when `MAX_RT` or `TX_DS` is high. To turn off the `IRQ` pin, reset the interrupt source by writing to the `STATUS` register (see Interrupt chapter). If no ACK packet is received for a packet after the maximum number of retransmits, no further packets can be transmitted before the `MAX_RT` interrupt is cleared. The packet loss counter (`PLOS_CNT`) is incremented at each `MAX_RT` interrupt. That is, `ARC_CNT` counts the number of retransmits that were required to get a single packet through. `PLOS_CNT` counts the number of packets that did not get through after the maximum number of retransmits.
6. nRF24L01+ goes into standby-I mode if `CE` is low. Otherwise, next payload in TX FIFO is transmitted. If TX FIFO is empty and `CE` is still high, nRF24L01+ enters standby-II mode.
7. If nRF24L01+ is in standby-II mode, it goes to standby-I mode immediately if `CE` is set low.

Enhanced ShockBurst™ receive payload

1. Select RX by setting the `PRIM_RX` bit in the `CONFIG` register to high. All data pipes that receive data must be enabled (`EN_RXADDR` register), enable auto acknowledgement for all pipes running Enhanced ShockBurst™ (`EN_AA` register), and set the correct payload widths (`RX_PW_Px` registers). Set up addresses as described in item 2 in the Enhanced ShockBurst™ transmitting payload example above.
2. Start Active RX mode by setting `CE` high.
3. After 130µs nRF24L01+ monitors the air for incoming communication.
4. When a valid packet is received (matching address and correct CRC), the payload is stored in the RX-FIFO, and the `RX_DR` bit in `STATUS` register is set high. The `IRQ` pin is active when `RX_DR` is high. `RX_P_NO` in `STATUS` register indicates what data pipe the payload has been received in.
5. If auto acknowledgement is enabled, an ACK packet is transmitted back, unless the `NO_ACK` bit is set in the received packet. If there is a payload in the `TX_PLD` FIFO, this payload is added to the ACK packet.
6. MCU sets the `CE` pin low to enter standby-I mode (low current mode).
7. MCU can clock out the payload data at a suitable rate through the SPI.
8. nRF24L01+ is now ready for entering TX or RX mode or power down mode.

Appendix B - Configuration for compatibility with nRF24XX

How to setup nRF24L01+ to receive from an nRF2401/nRF2402/nRF24E1/nRF24E2:

1. Use the same CRC configuration as the nRF2401/nRF2402/nRF24E1/nRF24E2.
2. Set the `PWR_UP` and `PRIM_RX` bit to 1.
3. Disable auto acknowledgement on the data pipe that is addressed.
4. Use the same address width as the PTX device.
5. Use the same frequency channel as the PTX device.
6. Select data rate 1Mbps or 250kbps on both nRF24L01+ and nRF2401/nRF2402/nRF24E1/nRF24E2.
7. Set correct payload width on the data pipe that is addressed.
8. Set `CE` high.

How to setup nRF24L01+ to transmit to an nRF2401/nRF24E1:

1. Use the same CRC configuration as the nRF2401/nRF2402/nRF24E1/nRF24E2.
2. Set the `PRIM_RX` bit to 0.
3. Set the Auto Retransmit Count to 0 to disable the auto retransmit functionality.
4. Use the same address width as the nRF2401/nRF2402/nRF24E1/nRF24E2.
5. Use the same frequency channel as the nRF2401/nRF2402/nRF24E1/nRF24E2.
6. Select data rate 1Mbps or 250kbps on both nRF24L01+ and nRF2401/nRF2402/nRF24E1/nRF24E2.
7. Set `PWR_UP` high.
8. Clock in a payload that has the same length as the nRF2401/nRF2402/nRF24E1/nRF24E2 is configured to receive.
9. Pulse `CE` to transmit the packet.

Appendix C - Constant carrier wave output for testing

The output power of a radio is a critical factor for achieving wanted range. Output power is also the first test criteria needed to qualify for all telecommunication regulations.

Configuration

1. Set `PWR_UP = 1` and `PRIM_RX = 0` in the `CONFIG` register.
2. Wait 1.5ms `PWR_UP`->standby.
3. In the RF register set:
 - ▶ `CONT_WAVE = 1`.
 - ▶ `PLL_LOCK = 1`.
 - ▶ `RF_PWR`.
4. Set the wanted RF channel.
5. Set `CE` high.
6. Keep `CE` high as long as the carrier is needed.

Note: Do not use `REUSE_TX_PL` together with `CONT_WAVE=1`. When both these registers are set the chip does not react when setting `CE` low. If however, both registers are set `PWR_UP = 0` will turn TX mode off.

The nRF24L01+ should now output an unmodulated centered carrier.

TECHNICAL DATA

MQ-3 GAS SENSOR

FEATURES

- * High sensitivity to alcohol and small sensitivity to Benzine .
- * Fast response and High sensitivity
- * Stable and long life
- * Simple drive circuit

APPLICATION

They are suitable for alcohol checker, Breathalyser.

SPECIFICATIONS

A. Standard work condition

Symbol	Parameter name	Technical condition	Remarks
V _c	Circuit voltage	5V±0.1	AC OR DC
V _H	Heating voltage	5V±0.1	AC OR DC
R _L	Load resistance	200K Ω	
R _H	Heater resistance	33 Ω ± 5%	Room Tem
P _H	Heating consumption	less than 750mw	

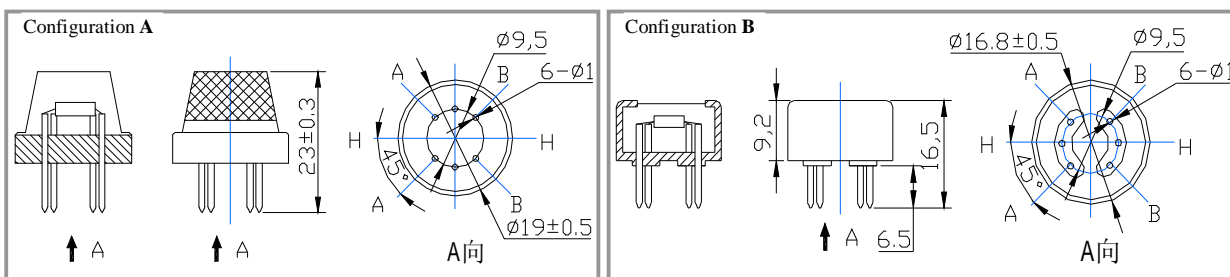
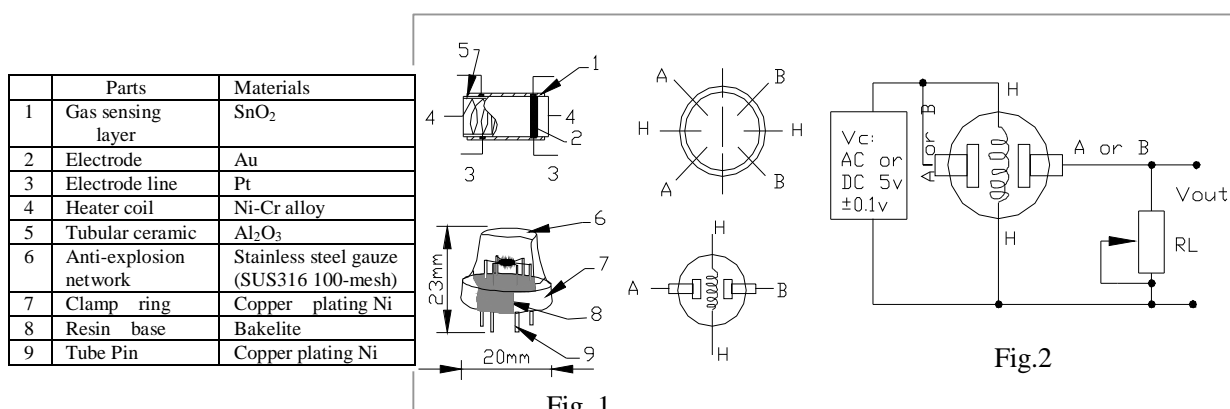
B. Environment condition

Symbol	Parameter name	Technical condition	Remarks
T _{ao}	Using Tem	-10℃-50℃	minimum value is over 2%
T _{as}	Storage Tem	-20℃-70℃	
R _H	Related humidity	less than 95% Rh	
O ₂	Oxygen concentration	21%(standard condition)Oxygen concentration can affect sensitivity	

C. Sensitivity characteristic

Symbol	Parameter name	Technical parameter	Remarks
Rs	Sensing Resistance	1MΩ - 8 MΩ (0.4mg/L alcohol)	Detecting concentration scope: 0.05mg/L—10mg/L Alcohol
α (0.4/1 mg/L)	Concentration slope rate	≤0.6	
Standard detecting condition	Temp: 20℃±2℃ Humidity: 65%±5%	Vc:5V±0.1 Vh: 5V±0.1	
Preheat time	Over 24 hour		

D. Structure and configuration, basic measuring circuit



Structure and configuration of MQ-3 gas sensor is shown as Fig. 1 (Configuration A or B), sensor composed by micro Al_2O_3 ceramic tube, Tin Dioxide (SnO_2) sensitive layer, measuring electrode and heater are fixed into a crust made by plastic and stainless steel net. The heater provides necessary work conditions for work of sensitive components. The enveloped MQ-3 have 6 pin ,4 of them are used to fetch signals, and other 2 are used for providing heating current.

Electric parameter measurement circuit is shown as Fig.2

E. Sensitivity characteristic curve

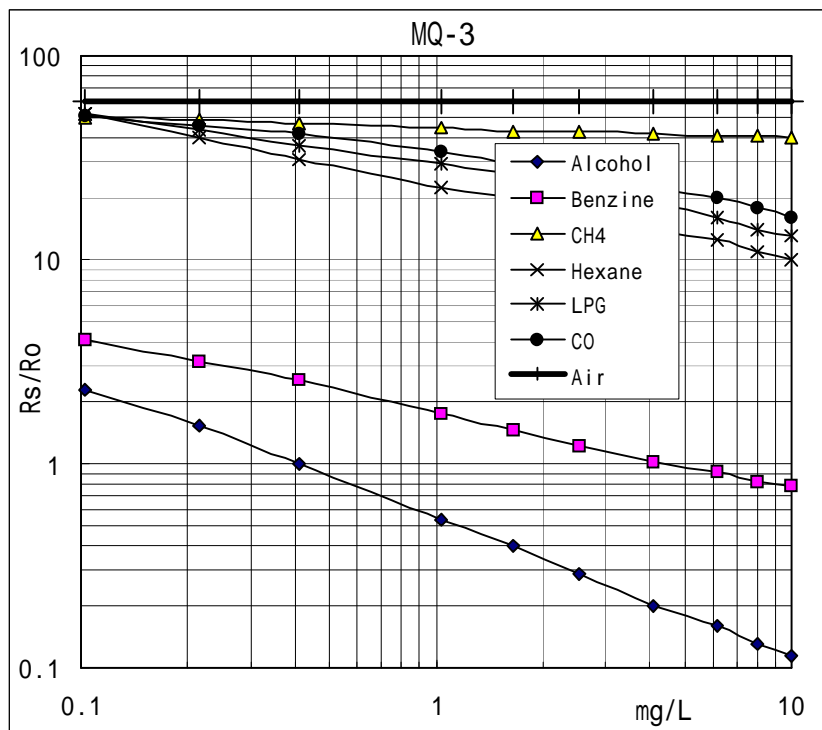


Fig.2 sensitivity characteristics of the MQ-3

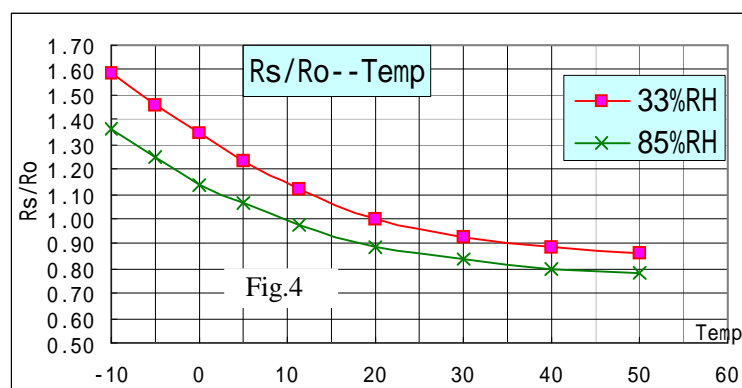


Fig.3 is shows the typical sensitivity characteristics of the MQ-3 for several gases.

in their: Temp: 20°C、

Humidity: 65%、

O₂ concentration 21%

RL=200k Ω

Ro: sensor resistance at 0.4mg/L of Alcohol in the clean air.

Rs:sensor resistance at various concentrations of gases.

Fig.4 is shows the typical dependence of the MQ-3 on temperature and humidity.

Ro: sensor resistance at 0.4mg/L of Alcohol in air at 33%RH and 20 °C

Rs: sensor resistance at 0.4mg/L of Alcohol at different temperatures and humidities.

SENSITIVITY ADJUSTMENT

Resistance value of MQ-3 is difference to various kinds and various concentration gases. So,When using this components, sensitivity adjustment is very necessary. we recommend that you calibrate the detector for 0.4mg/L (approximately 200ppm) of Alcohol concentration in air and use value of Load resistance that(R_L) about 200 K Ω (100K Ω to 470 K Ω).

When accurately measuring, the proper alarm point for the gas detector should be determined after considering the temperature and humidity influence.

ME 445: Final Project

Sound Following Hovercraft

Submitted by:
Henry Velasquez
Daniel Peterson
12/17/14

Table of Contents

Introduction	2
Acknowledgments	2
Bill of Materials	2
Mechanical Setup	3
Electronic Setup	3
Power	3
Speed Control	4
Microphone Implementation	4
Audio Signal Manipulation	4
Arduino Programming	5
Motor Control	5
Measuring Amplitude	5
PD Control	6
Testing and Calibration	6
Conclusion	6
Appendix	7
Programming Flow Chart	7
Code	8
Schematic	11
LM 7805 Datasheet	12
Electret Microphone Breakout Datasheet	13

Introduction

The objective of this project was to create a sound following hovercraft that will orient and move itself toward sound at a specific frequency. This is a spinoff on the traditional line follower and poses more of a challenge in terms of the signal manipulation and control. Brushless motors were used for thrust and lift and a servo was used for controlling the yaw of the hovercraft through rudder angle adjustment. Sound signals were obtained by the use of electret microphones. PD control was implemented in order to control the rudder to a high degree of accuracy. This report provides the details of our design and the process of developing our project.

Acknowledgements

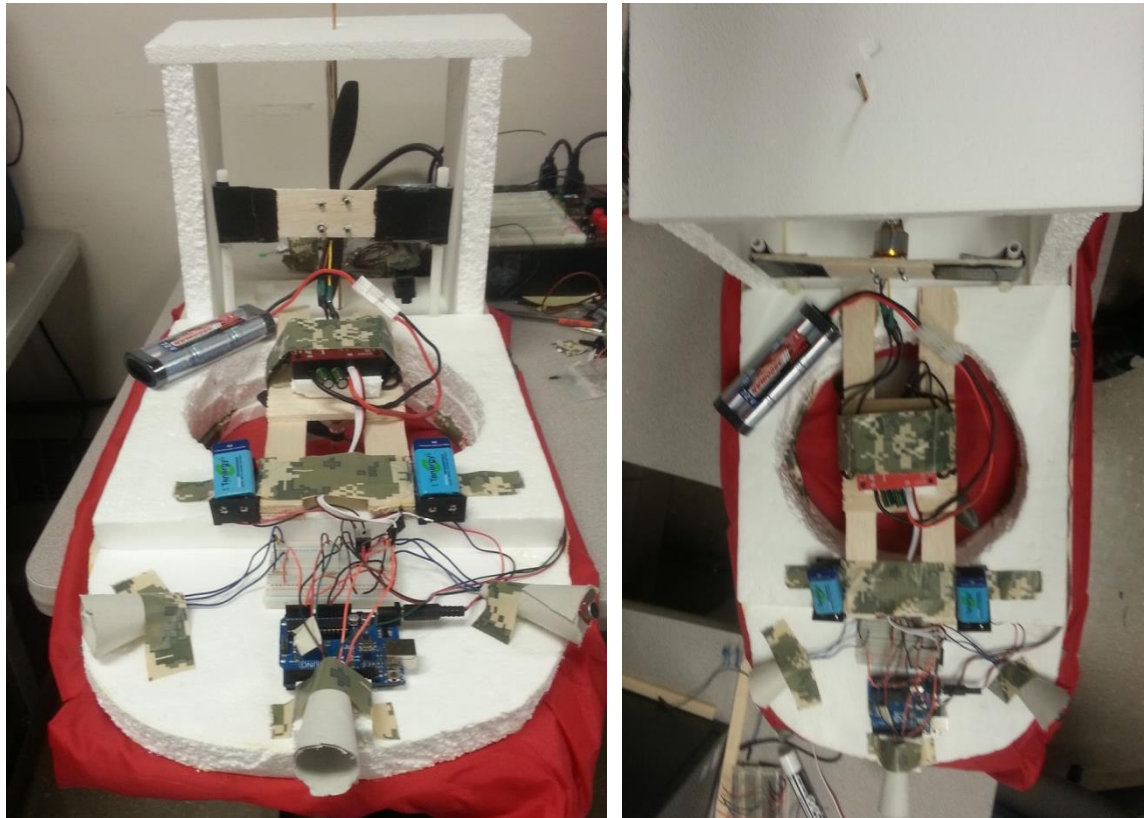
We would like to thank Dr. Sommer and Michael Robinson for providing the tools and information that allowed us to create this project. As mechanical engineers, we have spent much more time working with purely mechanical or fluid systems than we have with programming or electronics. It is important to realize that in real life, many of the projects/products we will be working on will concern electronics for mechanical control. This class helped us to become more comfortable and familiar electrical systems and tools.

Bill of Materials

Part	Quantity
Arduino	1
Brushless DC Motors	2
Futuba S3003 Servo	1
Sparkfun Electret Microphones	3
9V Tenenergy Battery	2
Balsa Wood	~
Styrofoam Boards (1'x1')	3
Rip-Stop Nylon	~
.015 μ F Capacitors	3
1k Ω Resistors	3
Traxxas 11.1V Battery 2100C	1
LM7805CT Voltage Regulator	1
E-MAX 4in1 Electronic Speed Controller	1

Mechanical Setup

All mechanical components of the Arduino hovercraft are supported by a strong Styrofoam and balsa wood body. The hovercraft gains movement by the use of 2 brushless DC motors, and 1 servo motor. One of the DC motors is designated for lift of the hovercraft and the other is designated for thrust. A Styrofoam housing was assembled around the thrust motor in order help direct the air backwards. Air accumulates in a rip stop nylon skirt beneath the hovercraft from the lift motor. The skirt has been strategically cut to have release some air so that the hovering is level and balanced. The servo is designated for turning the hovercraft by controlling the angle of the rudder within a 120 degree range of motion. The servo position is determined by the sound sensors at the front of the hovercraft.



Figures 1&2: Front and side views of completed hovercraft.

Electronics Setup

Power

The Arduino PIC is powered by a 9V battery. This is done using the stock 9V jack mounted on the Arduino. The Arduino then provides power to each of the sound sensors and their filters. Attempting to power the servo motor with the same power would cause issues due to lack of current, so a second 9V battery was dedicated to powering the servo as well as the electronic speed controller. The servo and ESC are only rated for a maximum 6 and 5.5 volts respectively. To avoid damaging the servo and ESC, we used a 7805CT voltage regulator which converted the 9V to the safe 5V energy source that we needed. A separate 11.1V LiPo battery was used to power the brushless motors through the ESC.

Speed Control

The speeds of the motors are controlled through a 4 in 1 EMAX electronic speed controller. Only 2 of the controllers are used; one for the thrust motor and one for the lift motor. The Arduino then sends signals to the ESC which tells the motors how fast to spin. This value must be within a range from 1000 to 2000, with 2000 being full speed. The DC motors are connected to the ESC through a 3-wire connection.

Microphone Implementation

The electret microphones require a 5V power source. All three microphones are connected to the Arduino power source in parallel. At first it was unclear what signal we were receiving from the microphones, as it was falsely assumed that they measured the amplitude of incoming sound. Through some research and experimentation, we found out that the microphones were giving us a sine wave with values between 0 and 700. In order to manipulate this signal we had to create first order high pass analog filters. For a cutoff frequency of about 10000 Hz we used a 1k Ω resistor and a .015 μ F capacitor. Each microphone got its own filter and sent its signals through the filter before reaching the Arduino analog pins. We will go into description of the signal manipulation in the next section.

Audio Signal Manipulation

The hovercraft alone makes a large amount of noise. Our mission was to figure out a way to make the microphones only respond to signals from a 10000Hz or higher frequency tone produced by a cellphone frequency generator app. As mentioned before, we used first order high-pass filters to achieve this. Although our filters still allowed some unwanted noise from the motors, the noise was attenuated enough for the Arduino to ignore them.

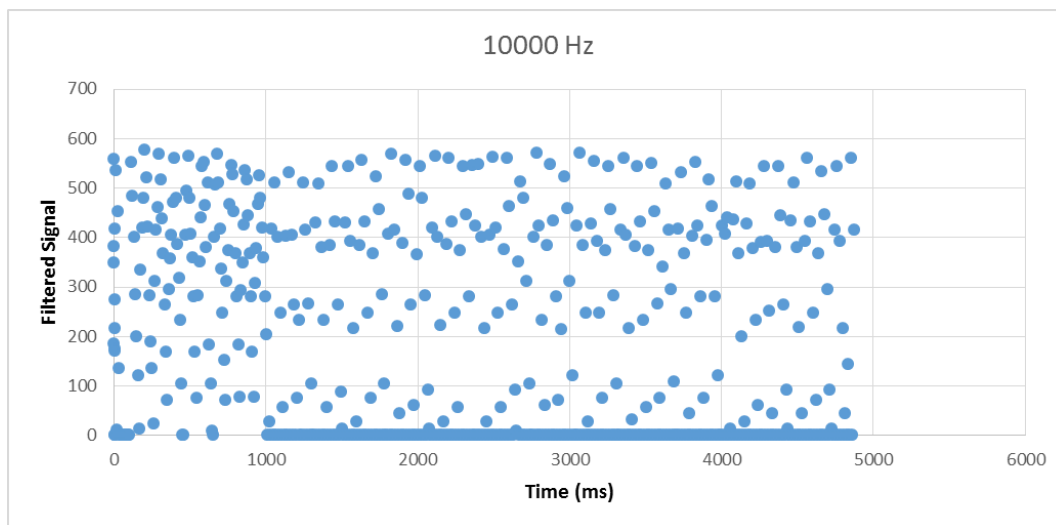


Figure 3: Filtered 10000 Hz signal.

The figure above shows the signal received from one of the sound sensors while playing a sound at 10000 Hz. It is clear that nearly the full range of the sound is picked up with little attenuation. This is the frequency we used to control the motion of the hovercraft.

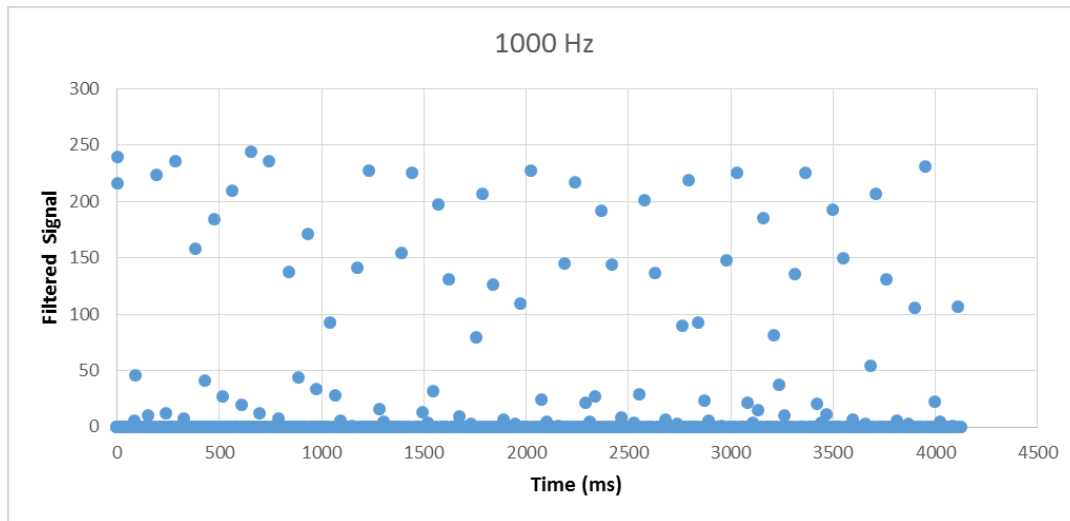


Figure 4: Filtered 1000 Hz signal.

The figure above shows a 1000 Hz signal passing through the same filter. Because it is significantly less than the cutoff frequency, its amplitude was diminished substantially. By reducing the amplitudes of everything below the frequency we used for control, it was much easier for the Arduino to pick out what sensor was picking up the signal. The programming behind this process is elaborated on in the following section.

Arduino Programming

The code for this project accomplished three major tasks: controlling the brushless motors, measuring the amplitude of the incoming signal, and controlling the rudder through PD control. A flowchart of the entire process can be found in the appendix.

Motor Control

The brushless motors that were used for lift and thrust required arming through the electronic speed controller before they could be used. This process required a signal to be sent to the ESC with the maximum speed, followed by another signal with the minimum speed a few seconds later. After the motors were armed, the thrust motor and lift motor could be engaged or disengaged through separate functions. For our purposes the motors were kept at a constant speed.

Measuring Amplitude

In order for the hovercraft to follow sound, the amplitudes from each sensor had to be recorded. This was an issue however because the incoming sound signal was a sine wave rather than the amplitude or sound pressure. In order to move forward the signals needed to be converted into amplitude so that they could later be compared with each other. In order to accomplish this, the Arduino was programmed to take thirty samples of each of the incoming signals, then to only record the maximum sample of each for comparison. These samples are taken quickly enough that it does not cause a noticeable lag in response time.

PD Control

PD control was used to control the motion of hovercraft so that it would follow sound generated at a high frequency. We excluded the integral aspect of PID control in order to avoid phase lag problems and to increase the response time. In order to use PD control, the proportional error first had to be calculated by comparing the amplitudes from the left and right sound sensors. Through testing we found that if the sound was held at a reasonable distance in front of the sensors, the proportional error would range from about -100 to 100. The positive values occurred when the sound was coming from the right of the hovercraft, and the negative values occurred when it was on the left. The range of our rudder was from about 30 to 150 degrees, so we decided to apply a gain of .6 to our proportional error. This scaled it down to a range of -60 to 60, which when added to the rudder midpoint of 90, would result in the appropriate range of motion. The derivative error was calculated by taking the change in error over the change in time. During testing this error ranged from -2 to about 2 while moving the sound at a reasonable speed. A gain of 2 was applied to this error so that it would have a slightly higher effect. This value was then subtracted from the angle calculated with proportional gain alone in order to reduce some of the slight overshoot we were seeing. After the angle was adjusted it was sent to the servo controlling the rudder angle.

Testing and Calibration

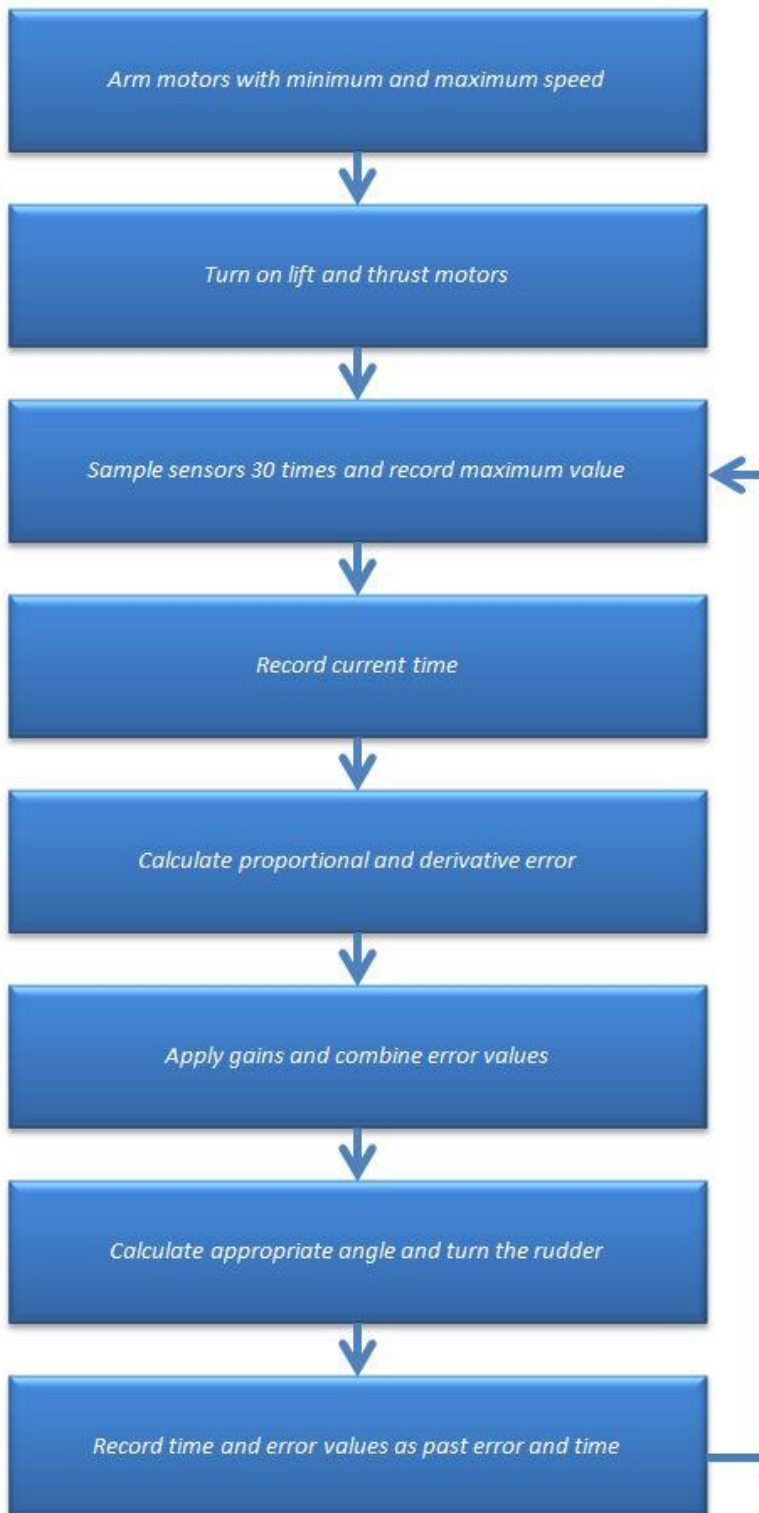
Upon completion of the prototype assembly, a few problems came up. One problem was turning radius. Our rudder control was working properly but we could not get the hovercraft to turn hard enough. We calibrated the max turning angles in the code so that the hovercraft would turn towards the sound quicker, but the main fix was increasing the rudders area. This caused more of a momentum shift via the amount of air hitting the rudder. Another problem that arose was that our thrust speeds varied based on our LiPo battery levels. This added more testing time since we were always unsure about how our hovercraft would react to the battery. In the future, we could fix this issue by adding a 3rd sensing mechanism that would regulate the speed. As explained before, a lot of time went into calibrating our filters so that our sound control would work efficiently.

Conclusion

In the end the hovercraft performed as we had anticipated. The hovercraft was able to identify the direction of the incoming sound, and adjust the rudder through PD control in order to reorient itself to follow the noise as it moved forward. The analog filters that we implemented successfully reduced much of the background noise as well as the noise generated by the motors themselves. If we were able to continue on this project, we would have the thrust motor vary its speed based on distance it is from the high frequency sound. Although the thrust motor was not spinning fast enough during the demonstration, we were able to navigate around the desks in the room previously to make a loop. Overall we are very satisfied with this project, and wish we had more time to develop it further.

Appendix

Programming Flow Chart



Code

```
//Include library of commands for servo control
#include <Servo.h>

//Sound amplitude variables
int ampF;    //variable for storing amplitude measured in front of the hovercraft
int ampL;    //variable for storing amplitude measured on the left of the hovercraft
int ampR;    //variable for storing amplitude measured on the right of the hovercraft
int f;  //temporary variable for finding the highest amplitude in front of the hovercraft
int r;  //temporary variable for finding the highest amplitude on the left of the hovercraft
int l;  //temporary variable for finding the highest amplitude on the right of the hovercraft

//PID control variables
int error = 0;    //measure how far off the hovercraft is from facing the noise
float derivative; //stores the derivative of the error
int pastError = 0; //stores the error from the previous pass
int time = 0;     //stores the time that amplitude was recorded
int pastTime = 0; //stores the time from the previous pass
float gain = .6;  //gain for the error, .6 chosen to scale the error down to a range of 30
                  //degrees to 120 degrees for rudder control
float dgain = 2;  //gain for the derivative error, 2 chosen to scale the derivative error up

//Rudder variables
int angle;    //angle to write to rudder servo
int midpoint = 90; //midpoint position for rudder to start and return to
int rudderPin = 9; //output pin for rudder servo
Servo rudder;   //rudder servo control variable

//Motors
Servo liftESC;  //lift motor control variable, motor4 on ESC
Servo thrustESC; //thrust motor control variable, motor2 on ESC
int liftPin = 10; //Output arduino pin for lift motor
int thrustPin = 11; //Output arduino pin for thrust motor

void setup()
{
  Serial.begin(9600);    //connect with serial monitor

  liftESC.attach(liftPin); //attach servo to liftPin
  thrustESC.attach(thrustPin); //attach servo to thrustPin

  rudder.attach(rudderPin); //connect servo to rudderPin
  rudder.write(midpoint);   //turn servo to mid-point

  arm();//call function to arm the ESC
}
```

```

void loop()
{
  lift(true);    //turn on lift motor
  thrust(true); //turn on thrust motor

  //Set amplitudes to 0 before recording new set
  ampF = 0;
  ampR = 0;
  ampL = 0;

  //loop to select the highest amplitude recorded over 20 samples
  for(int i = 0; i < 20; i++)
  {
    l=analogRead(0); //take sample from left sound sensor
    f=analogRead(1); //take sample from front sound sensor
    r=analogRead(2); //take sample from right sound sensor
    if(f > ampF) //if current sample is greater than previous samples
      ampF = f; //set amplitude variable to new higher value
    if(r > ampR)
      ampR = r;
    if(l > ampL)
      ampL = l;
  }
  time = millis();          //record current time

  //error is how far off the sound is from being directly in front of the hovercraft
  //we calculate error by measuring the difference in amplitude on the left and right sides
  //zero error is recorded if both the left and right sensors record the same amplitude
  error = ampR - ampL;
  //derivative error is calculated as the change in error over the change in time
  derivative = (float)(error-pastError)/(float)(time-pastTime);
  //angle for the servo is found by combining the proportional error and derivative error
  angle=(gain*error)+midpoint - (dgain*derivative);

  //upper and lower limits set to prevent the servo from over turning
  if(angle >145)
    angle = 145;
  if(angle < 35)
    angle = 35;

  rudder.write(angle);      //turn servo to computed angle

  //set past error and time
  pastError = error;
  pastTime = time;
}
//function to turn rudder, currently unused

```

```

void turn(int angle)
{
  angle=(midpoint+angle);    //new angle in relation to midpoint
  //Serial.println(angle);
  rudder.write(angle);      //turn servo angle degrees
}

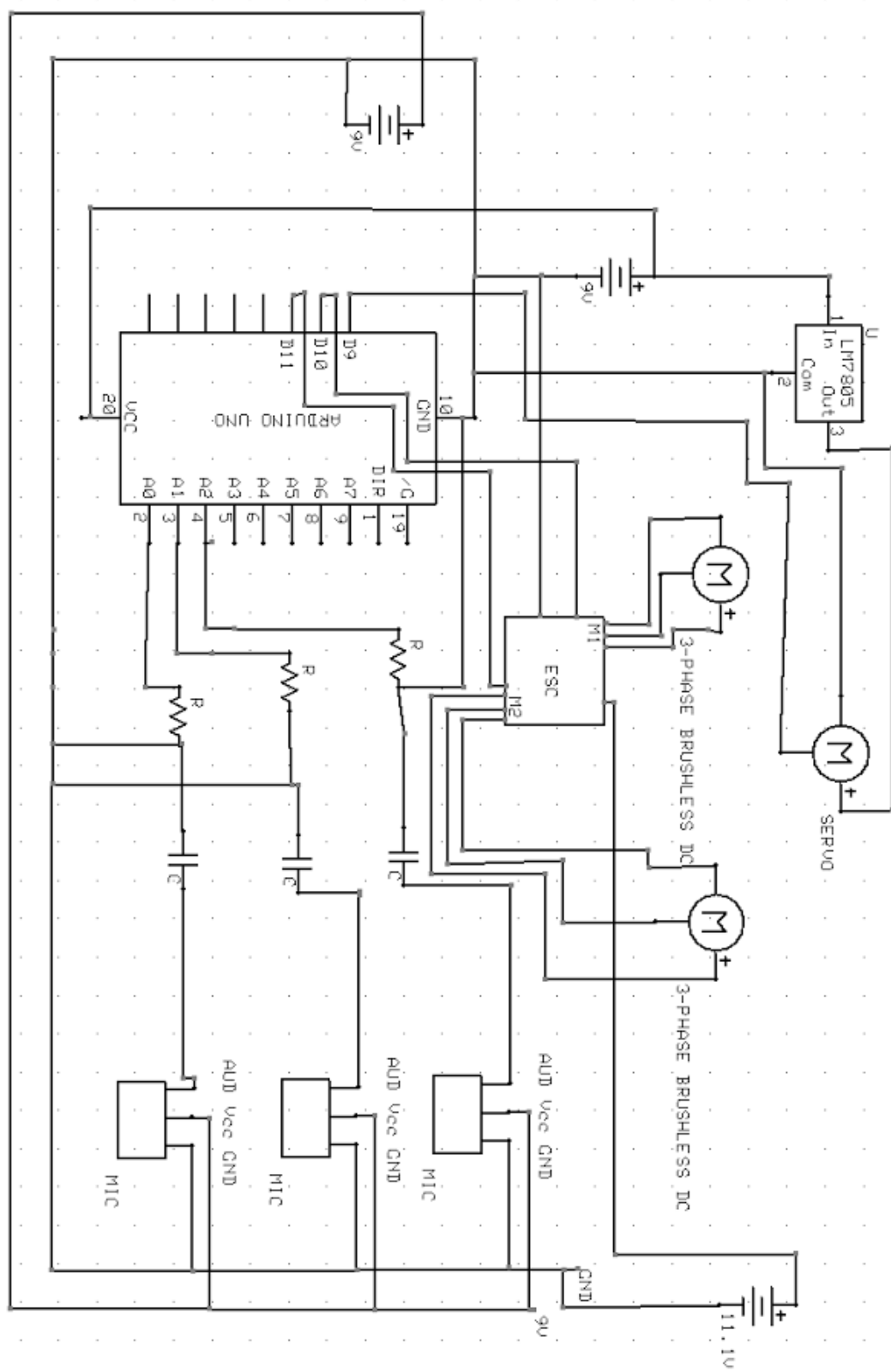
//function to arm the esc for use
void arm()
{
  delay(6000);              //wait for esc to be programmed
  thrustESC.writeMicroseconds(1500); //set high thrust speed
  liftESC.writeMicroseconds(2000); //set high lift speed
  delay(4000);
  thrustESC.writeMicroseconds(1000); //set low thrust speed
  liftESC.writeMicroseconds(1000); //set low lift speed
  delay(100);
}

//function to turn thrust motor on and off
void thrust(boolean on)
{
  if(on)
  {
    thrustESC.writeMicroseconds(1400); //full speed
  }
  else
  {
    thrustESC.writeMicroseconds(0);    //off
  }
}

//function to turn lift motor on and off
void lift(boolean on)
{
  if(on)
  {
    liftESC.writeMicroseconds(2000); //full speed
  }
  else
  {
    liftESC.writeMicroseconds(0);      //off
  }
}

```

Schematic



LM7805 Datasheet

Electrical Characteristics (LM7805A)

Refer to the test circuit, $0^{\circ}\text{C} < T_J < 125^{\circ}\text{C}$, $I_O = 1\text{ A}$, $V_I = 10\text{ V}$, $C_I = 0.33\text{ }\mu\text{F}$, $C_O = 0.1\text{ }\mu\text{F}$, unless otherwise specified.

Symbol	Parameter	Conditions	Min.	Typ.	Max.	Unit
V_O	Output Voltage	$T_J = +25^{\circ}\text{C}$	4.9	5.0	5.1	V
		$I_O = 5\text{ mA to }1\text{ A}$, $P_O \leq 15\text{ W}$, $V_I = 7.5\text{ V to }20\text{ V}$	4.8	5.0	5.2	
Regline	Line Regulation ⁽²⁰⁾	$V_I = 7.5\text{ V to }25\text{ V}$, $I_O = 500\text{ mA}$		5.0	50.0	mV
		$V_I = 8\text{ V to }12\text{ V}$		3.0	50.0	
		$T_J = +25^{\circ}\text{C}$		5.0	50.0	
				1.5	25.0	
Regload	Load Regulation ⁽²⁰⁾	$T_J = +25^{\circ}\text{C}$, $I_O = 5\text{ mA to }1.5\text{ A}$		9	100	mV
		$I_O = 5\text{ mA to }1\text{ A}$		9	100	
		$I_O = 250\text{ mA to }750\text{ mA}$		4	50	
I_Q	Quiescent Current	$T_J = +25^{\circ}\text{C}$		5	6	mA
ΔI_Q	Quiescent Current Change	$I_O = 5\text{ mA to }1\text{ A}$			0.5	mA
		$V_I = 8\text{ V to }25\text{ V}$, $I_O = 500\text{ mA}$			0.8	
		$V_I = 7.5\text{ V to }20\text{ V}$, $T_J = +25^{\circ}\text{C}$			0.8	
$\Delta V_O / \Delta T$	Output Voltage Drift ⁽²¹⁾	$I_O = 5\text{ mA}$		-0.8		mV/ $^{\circ}\text{C}$
V_N	Output Noise Voltage	$f = 10\text{ Hz to }100\text{ kHz}$, $T_A = +25^{\circ}\text{C}$		42		μV
RR	Ripple Rejection ⁽²¹⁾	$f = 120\text{ Hz}$, $V_O = 500\text{ mA}$, $V_I = 8\text{ V to }18\text{ V}$		68		dB
V_{DROP}	Dropout Voltage	$I_O = 1\text{ A}$, $T_J = +25^{\circ}\text{C}$		2		V
R_O	Output Resistance ⁽²¹⁾	$f = 1\text{ kHz}$		17		m Ω
I_{SC}	Short-Circuit Current	$V_I = 35\text{ V}$, $T_J = +25^{\circ}\text{C}$		250		mA
I_{PK}	Peak Current ⁽²¹⁾	$T_J = +25^{\circ}\text{C}$		2.2		A

Notes:

20. Load and line regulation are specified at constant junction temperature. Changes in V_O due to heating effects must be taken into account separately. Pulse testing with low duty is used.

21. These parameters, although guaranteed, are not 100% tested in production.

**Challenge Electronics**

95 East Jeffryn Boulevard

Deer Park, NY 11729

EMAIL: SALES@CHALLELEC.COM

Tel: 1-800-722-8197

1-631-595-2217

Fax: 1-631-586-5899

WEB: WWW.CHALLENGEELECTRONICS.COM

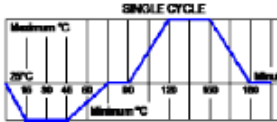
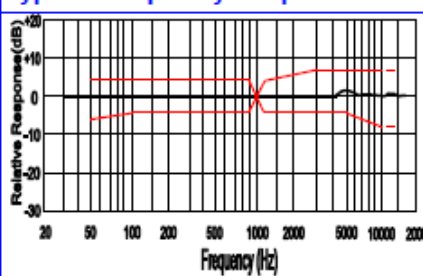
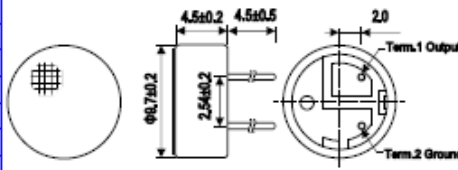
• ISO 9001:2000

• ISO 14001:2004

• ISO/TS 16949:2002



PRODUCT INFORMATION

PART #		CEM-C9745JAD462P2.54R				Revision		0-2010																															
Omni-Directional Foil Electret Condenser Microphone																																							
DESCRIPTION																																							
Omni-Directional Foil Electret Microphone, 9.7 mm diameter and 4.5 mm high, Power Supply 5.0 V max, External Resistance Loading of 680 Ω , and sensitivity of -44 dB. Terminated with 2 solder points, Lead Free RoHS Compliant																																							
SPECIFICATIONS:																																							
Direction		Omni Directional Foil Electret			Minimum Direction sensitivity																																		
Operating Voltage Range		Vs = 1.0 Vdc ~ 10.0 Vdc			Power Supply (Vs)		1.5 V																																
Frequency Range		100 ~ 10,000 Hz.			Maximum Current		0.5 mA																																
Sensitivity		-46 \pm 2.0, (0 dB = 1V / Pa) at 1K Hz.			Minimum Sensitivity to Noise Ratio		58 dB																																
Sensitivity Reduction		3.0 V to 2.0 V -3 dB			Maximum input S.P.L.		110 dB at 1.0 KHz, THD <1%																																
Operating Temperature		-20°C to + 60°C			Storage Temperature		-40°C to + 75°C																																
Loading Resistance (RL)		External, 680 Ω at Vs = 1.5 V, Max. 2,200 Ω			Built in Capacitors		None																																
Termination		PC Pins, 4.5 mm Long, 0.6 mm ϕ , 2.54 mm Spacing																																					
Dimensions		Length / Diameter		9.7 mm ϕ		Height		4.5 mm																															
Approximate Weight		0.7 grams		Options		Al-Mg Alloy.		Color																															
						Compliance		RoHS, Lead Free																															
Reliability																																							
Thermal Operating Cycle Test		250 hours continuous operation at Rated Power, at Maximum Rated Operating Temperature *																																					
		250 hours continuous operation at Rated Power, at Minimum Rated Operating Temperature *																																					
Thermal Storage Cycle Test		Parts are subjected to 250 hours storage at Maximum Rated Storage Temperatures *																																					
		Parts are subjected to 250 hours storage at Minimum Rated Storage Temperatures *																																					
Thermal Shock Test:		Parts are subjected to five (5) cycles of Minimum and Maximum Operating Temperature. Each cycle shall be set per diagram below and is three (3) hours long *																																					
Humidity Test		Parts are subjected to 240 Hours at +40°C \pm 2°C, 90-95% RH *																																					
Vibration Test		Parts are subjected to 2 Hours of at 1.5 mm with 10 to 55 Hz. vibration frequency to each of 3 perpendicular directions *																																					
Drop Test		Parts are dropped naturally from 1 meter height onto the surface of 40 mm wooden board, 2 axes (X,Y) directions, 3 times (6 times total) *																																					
Reliability Test Performance *		Parts should conform to original performance within \pm 5 dB tested with Rated Power, after 3 hours of recovery period.																																					
Termination Strength		Terminals should withstand a 1.0 Kg. pull test for up to 1 minute.																																					
Life Test		At rated voltage in room temperature continuously for 1,000 hours																																					
Warranty		For a period of one (1) year from date of shipping under normal operations conditions																																					
Typical Frequency Response				Dimensions Units in: mm Tolerance: \pm 0.3 mm																																			
				<table><tr><th colspan="3">Microphone Response Toller Window</th></tr><tr><th>Frequency (Hz)</th><th>Lower Limit (dB)</th><th>Upper Limit (dB)</th></tr><tr><td>50</td><td>-6</td><td>+3</td></tr><tr><td>100</td><td>-3</td><td>+3</td></tr><tr><td>800</td><td>-3</td><td>+3</td></tr><tr><td>1000</td><td>0</td><td>0</td></tr><tr><td>1200</td><td>-3</td><td>+3</td></tr><tr><td>3000</td><td>-3</td><td>+8</td></tr><tr><td>5000</td><td>-3</td><td>+8</td></tr><tr><td>10000</td><td>-8</td><td>+8</td></tr></table> 						Microphone Response Toller Window			Frequency (Hz)	Lower Limit (dB)	Upper Limit (dB)	50	-6	+3	100	-3	+3	800	-3	+3	1000	0	0	1200	-3	+3	3000	-3	+8	5000	-3	+8	10000	-8	+8
Microphone Response Toller Window																																							
Frequency (Hz)	Lower Limit (dB)	Upper Limit (dB)																																					
50	-6	+3																																					
100	-3	+3																																					
800	-3	+3																																					
1000	0	0																																					
1200	-3	+3																																					
3000	-3	+8																																					
5000	-3	+8																																					
10000	-8	+8																																					

The information contained herein is believed to be correct, but no guarantee or warranty, express or implied, with respect to accuracy, completeness or results is extended and no liability is assumed. Challenge Electronics reserves the right to make changes in any specification, data or material contained herein.

Copyright © 2010 Challenge Electronics

Quadcopter



ME 445 – Fall 2014

Andrew Wilson, Chris Pogash

Contents

Overview	3
Components	3
Mechanical	3
Quad Frame	3
Brushless Motors	3
Propellers.....	3
Electrical	3
ESC	3
Transmitter and Reciever	3
LiPo Battery	4
6-DOF Pololu IMU	4
Arduino	4
Basic Flight Theory	4
Data Collected	5
Issues and Improvements.....	6
Issues	7
Time	7
Greater task than anticipated.....	7
Component failures	7
Ran into too many troubleshooting issues	7
Improvements	7
Run 2 stepper motors on test fixture	7
Use an IMU with magnetometer	7
Add battery buzzer onto the Arduino shield	7
Replace ESC.....	7
More research and practice is needed	7
Work out issues left in code	7
Parts List	8
Code.....	9

Overview

A quadcopter, or sometimes referred to as a quadrocopter, is a flight vehicle that consists of 4 motors configured around a central body holding the flight controlling hardware. In recent years, there has been a significant increase in the use of these devices for military, law enforcement, photography, and even local shipping.

With this growing interest in the use of quadcopters we decided to focus our project on creating a quadcopter that could achieve stable flight using an Arduino and several other flight controlling components which are further analyzed within this report. Also explained, is the code we created to communicate instructions via Arduino to the ESC and motors in order to maintain stable flight.

Components

Mechanical

Quad Frame: The frame and most of the other components we used were built from a previous semester. The frame was very useful in that it already contained predrilled locations for the brushless motors and other hardware.

Brushless Motors: 4 Brushless motors were essential to create the necessary thrust to stabilize the quadcopter. The motors were set to a plus configuration in which opposing motors spin in the same direction while adjacent motors spin in opposite directions. By individually controlling the speed of these motors we can control the motion of the quadcopter in any of its 6 degrees of freedom.

Propellers: 4 propellers are needed, one for each motor. Sizing of these propellers are standardized for various motor sizes and applications. They must also be balanced in order to reduce vibrations within the system.

Electrical

ESC: We used a 25A 4-in-1 ESC controller to consolidate the hardware. This ESC could send the appropriate signals to each of the 4 motors to control individual rpm. Unfortunately this ESC failed to output signals for one of our motors and would not allow us to conduct proper testing.

Transmitter and Receiver: A typical hobby kit is all that is necessary to allow for human input via radio communication. The transmitter has the ability to control all six degrees of freedom and can adjust all controls to a zero, set point with trim. The receiver outputs on four channels including roll, pitch, yaw, and throttle.

LiPo Battery: A lithium-ion polymer (LiPo) battery was used to power the motors, Arduino, ESCs, and IMU. We used an 11.1V 2100mAh 20C LiPo battery for our application. These batteries allow for very large amounts of current to flow which should allow us to power all of these components for a flight time of around 15-20 minutes.

6-DOF IMU: We used an MPU-6050 which consisted of a 3 axis accelerometer and a 3 axis gyroscope. This IMU allowed us to read the flight dynamic data, to be used in conjunction with the PID control theory. The accelerometer and gyroscope data were fused together to give us projected angles on the x, y, and z axes corresponding to gravity.

Arduino: The Arduino Uno was the microprocessor that allowed us to communicate with all of the electronic components of our project.

Basic Flight Theory

- 6-DOF IMU

Utilizing the IMU from InvenSense we were able to gather acceleration and gyroscopic data for each axis. This data was handled by some simple trigonometry giving us all of the data necessary to control the flight of the quadcopter. It also provides us with data to use in our PID controls to eliminate noise and over/underacting to dynamics of the quadcopter. A closer look at the application of these computations can be seen in the code section of our report.

- Dynamics

The basic idea behind a quadcopter is to use the minimum number of motors in a plane to control all degrees of freedom for flight. The 6 degrees of freedom that we are using involves movement and rotation along each x, y, and z axis. We can manipulate all aspects of flight by applying different thrusts to the individual motors.

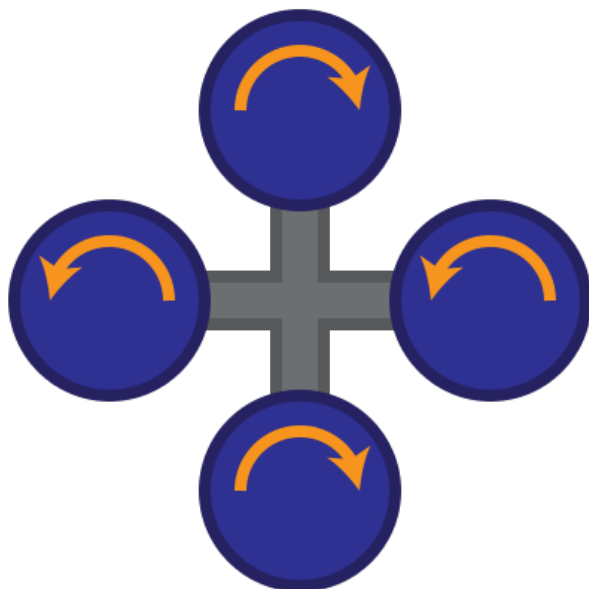


Figure 1: the orientation for stable flight is 4 motors with equal thrust. Adjacent motors spin in opposite directions while opposing motors spin in the same direction.

Yaw can be adjusted by increasing the speed of the two opposing motors with respect to the other pair of motors. This motion will rotate figure 1 in the plane of this paper.

The pitch and roll can also be adjusted by increasing the thrust on one motor while decreasing the thrust on the opposing motor. The other pair of motors must have equal thrust to achieve pure pitch and roll. This motion allows the quad to translate in the plane of this paper.

Data Collected

The test procedure we followed allows us to check the accuracy of our IMU with a known angle change. The test stand consists of a stepper motor coupled to a rotating rod that is secured to the quad frame itself. This stepper will rotate through 45 degrees in one direction then 45 degrees in the other and then back to a level position. We can compare this known information with the IMU readings.

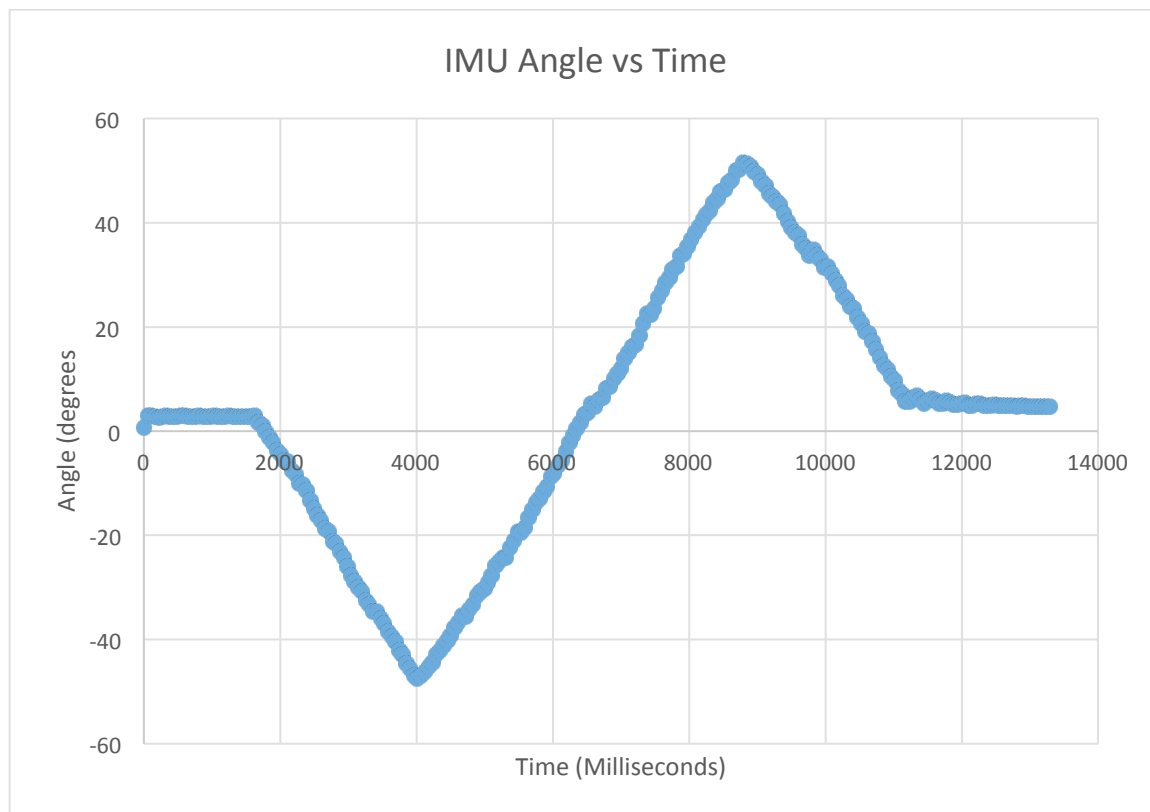


Figure 2: The IMU readings for rotation around the y-axis match the known application of angle change to

In addition to this testing procedure, we would recommend adding tachometers after the ESC has been replaced in order to gather motor speeds. This will aid in making adjustments to PID controls to achieve a stable flight. Below is a picture of our test stand.

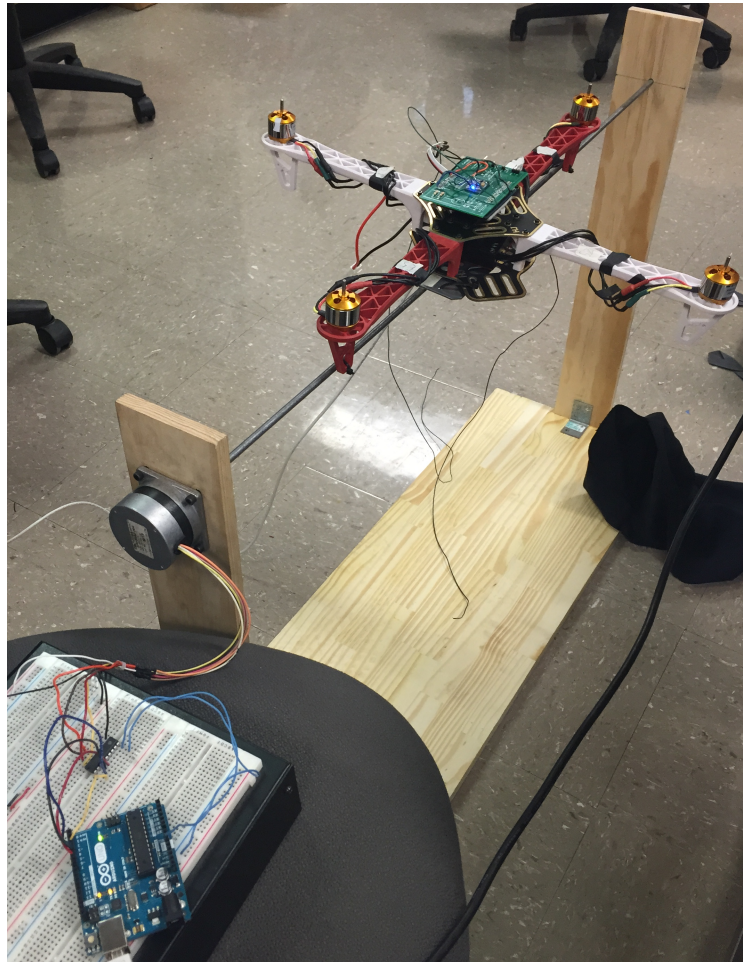


Figure 3: The testing fixture allows us to verify the angle the IMU communicates to the arduino. This also allows for testing and verification that the motors increase and decrease in velocity correctly. This adds a safety buffer as well since quads equipped with propellers can be quite dangerous.

Issues and Improvements

Issues

Time: The quadcopter project proved to be a very time intensive process which we could not foresee. Approximately 10 hours per week were spent working on the quad. For a successful project we should have spent closer to 20 hours per week. The 7-8 weeks that we had simply were not enough to have a fully operational quadcopter.

Greater task than anticipated: At the beginning of the semester we believed that the quadcopter was not as complex of a system as we anticipated. The complexity comes in learning how to integrate the various components to work together as a complete system. This required much new learning and training.

Component failures: In using an assembled quad from a previous semester we had to spend a great deal of time troubleshooting to find the root cause of operational issues. When it came time to test the motors we encountered a faulty ESC which forced us to abandon an attempt to test our quadcopter with all of our motors. After this we decided to use stepper motors to demonstrate the accuracy of our IMU but the stepper motor did not provide enough torque to hold the quad in place.

Ran into too many troubleshooting issues: There were far more troubleshooting issues with our project than we had originally anticipated. We faced many delays when trying to find the source of our setbacks. This caused major obstacles in meeting our goals.

Improvements

Run 2 stepper motors on test fixture: This will allow us to test the IMU accuracy on each axis without running into torque issues with the stepper.

Use an IMU with magnetometer (compass) for yaw control: As of now we have little control of yaw due to the fact that we do not have another reading specifically for our axis in the plane of the quad frame.

Add battery buzzer onto the Arduino shield: This will help us to avoid damaging our LiPo battery.

Replace ESC: We would like to replace the 4 in 1 ESC with individual ESCs so we do not have a difficult time replacing/finding an issue with our motors.

More research and practice is needed: The level of expertise to build a custom quad copter is beyond the ability we had coming into this project. It would be best to first design an RC car or plane and then apply that knowledge to a quadcopter build.

Work out issues left in code: Our code needs to be adjusted to the new components that have yet to be added and initializations need to be configured for the controller and other components.

Parts List

Part	Quantity	Price	Vendor
LiPo Battery	1	\$23.99	Amazon
IMU	1	\$39.95	Adafruit
Frame	1	\$19.95	Amazon
ESC	1	\$29.92	Hobby King
Brushless Motor	4	\$59.96	Hobby King
Transmitter/Receiver	1	\$59.99	Hobby King
Propeller kit	1	\$19.99	Amazon
Total		\$253.75	

This comprehensive parts list is the essential required material need to construct a basic quadcopter. In addition to these components the builder may want to design and order their own shield that will allow for the arduino to seamlessly communicate via I2C to the IMU and onward.

Code

```
// To Do List:
// Make sure all motors are hooked up and working properly (Is motor 2 plugged in correctly?)
// Which pins from ESC correspond to which motors? (Label them)
// Add controller_initialize() code - motors do not start until controller is turned on and
THROTTLE,PITCH,ROLL = 0
// Calculate angleX,Y,Z from AcX,Y,Z and GyX,Y,Z values in IMU code
// Add red and green leds and buzzer for battery status?
//

#include "Configuration.h"
#include <Math.h>
#include <PID_v1.h>
#include <PinChangeInt.h>
#include <Servo.h>
#include <Wire.h>

// Angles
float angleX,angleY,angleZ = 0.0;

// RX Signals
int throttle=THROTTLE_RMIN;
volatile int rx_values[4]; // ROLL, PITCH, THROTTLE, YAW

// PID variables
double pid_roll_in, pid_roll_out, pid_roll_setpoint = 0;
double pid_pitch_in, pid_pitch_out, pid_pitch_setpoint = 0;
double pid_yaw_in, pid_yaw_out, pid_yaw_setpoint = 0;

// Motors
int M1, M2, M3, M4; // Front, Right, Back, Left

// Track loop time.
unsigned long prev_time = 0;

void setup()
{
  #ifdef DEBUG_OUTPUT
    Serial.begin(115200);
    while(!Serial);
    Serial.println("Debug Output ON");
```

```

#endif

,

motors_initialize();
imu_initialize();
leds_initialize();
rx_initialize();
pid_initialize();
motors_arm();

//wait for IMU YAW to settle before beginning??? ~20s
}

void loop()
{
    imu_update();
    control_update();

#ifdef DEBUG_OUTPUT
    debug_process();
#endif
    prev_time = micros();
}

void controller_initialize(){
    while(1){
        if(PIN_RX_THROTTLE == THROTTLE_RMIN, PIN_RX_PITCH == PITCH_RMIN){
        }
    }
}

void control_update(){

throttle=map(rx_values[2],THROTTLE_RMIN,THROTTLE_RMAX,MOTOR_ZERO_LEVEL,
MOTOR_MAX_LEVEL);

    setpoint_update();
    pid_update();
    pid_compute();

```

```

// yaw control disabled for stabilization testing...
M1 = throttle + pid_pitch_out ;//+ pid_yaw_out;
M2 = throttle + pid_roll_out ;//- pid_yaw_out;
M3 = throttle - pid_pitch_out ;//+ pid_yaw_out;
M4 = throttle - pid_roll_out ;//- pid_yaw_out;

#ifdef SAFE
    if(throttle < THROTTLE_SAFE_SHUTOFF)
    {
        M1 = M2 = M3 = M4 = MOTOR_ZERO_LEVEL;
    }
#endif

update_motors(M1, M2, M3, M4);
}

void setpoint_update() {
    // here we allow +- 20 for noise and sensitivity on the RX controls...
    // ROLL rx at mid level?
    if(rx_values[0] > THROTTLE_RMID - 20 && rx_values[0] < THROTTLE_RMID + 20)
        pid_roll_setpoint = 0;
    else
        pid_roll_setpoint =
map(rx_values[0],ROLL_RMIN,ROLL_RMAX,ROLL_WMIN,ROLL_WMAX);
    //PITCH rx at mid level?
    if(rx_values[1] > THROTTLE_RMID - 20 && rx_values[1] < THROTTLE_RMID + 20)
        pid_pitch_setpoint = 0;
    else
        pid_pitch_setpoint =
map(rx_values[1],PITCH_RMIN,PITCH_RMAX,PITCH_WMIN,PITCH_WMAX);
    //YAW rx at mid level?
    if(rx_values[3] > THROTTLE_RMID - 20 && rx_values[3] < THROTTLE_RMID + 20)
        pid_yaw_setpoint = 0;
    else
        pid_yaw_setpoint =
map(rx_values[3],YAW_RMIN,YAW_RMAX,YAW_WMIN,YAW_WMAX);
}

#include <Wire.h>

const int MPU=0x68; // I2C address of the MPU-6050

```

```
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ; //sets all variables to 16 bit integers
```

```
void imu_initialize(){  
    Wire.begin();  
    Wire.beginTransmission(MPU);  
    Wire.write(0x6B); // PWR_MGMT_1 register  
    Wire.write(0);    // set to zero (wakes up the MPU-6050)  
    Wire.endTransmission(true);  
}
```

```
void imu_update()  
{  
    // Using an MPU-6050  
    // Contains: 3-axis accelerometer  
    //          3-axis gyroscope  
  
    // MPU-6050 orientation (positive direction):
```

```
    //   ^ y (towards M1, PITCH)  
    //   |  
    //   |  
    //   |___> x (towards M2, ROLL)  
    //   /  
    //  /  
    // z (towards sky, TROTTLER)  
    //  
    // Gyroscope positive value directions:  
    // x - pitch up, towards sky  
    // y - roll right, towards M2  
    // z - yaw left, towards M3
```

```
    Wire.beginTransmission(MPU);  
    Wire.write(0x3B); // starting with register 0x3B (ACCEL_XOUT_H)  
    Wire.endTransmission(false);  
    Wire.requestFrom(MPU,14,true); // request a total of 14 registers  
    int raw_AcX=Wire.read()<<8|Wire.read(); // 0x3B (ACCEL_XOUT_H) & 0x3C  
    (ACCEL_XOUT_L)  
    int raw_AcY=Wire.read()<<8|Wire.read(); // 0x3D (ACCEL_YOUT_H) & 0x3E  
    (ACCEL_YOUT_L)  
    int raw_AcZ=Wire.read()<<8|Wire.read(); // 0x3F (ACCEL_ZOUT_H) & 0x40  
    (ACCEL_ZOUT_L)  
    int Tmp=Wire.read()<<8|Wire.read(); // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
```

```

    int raw_GyX=Wire.read()<<8|Wire.read(); // 0x43 (GYRO_XOUT_H) & 0x44
(GYRO_XOUT_L)
    int raw_GyY=Wire.read()<<8|Wire.read(); // 0x45 (GYRO_YOUT_H) & 0x46
(GYRO_YOUT_L)
    int raw_GyZ=Wire.read()<<8|Wire.read(); // 0x47 (GYRO_ZOUT_H) & 0x48
(GYRO_ZOUT_L)


// Converts raw data to a voltage value
// Value = raw_value * (Vref / (LSB/g*2)) +- offset / sensitivity
// Accelerometer values in units g (9.81 m/s^2) (range is +-2g)
// Gyroscope values in units deg/s (range is +-250 deg/s)
float AcX = raw_AcX * (5.0 / 32767.0) - 0.1 / 2.5; //accelerometer sensitivity = 2.5V/g
float AcY = raw_AcY * (5.0 / 32767.0) - 0.07 / 2.5;
float AcZ = raw_AcZ * (5.0 / 32767.0) + 0.5 / 2.5;
float GyX = raw_GyX * (5.0 / 32767.0) / 0.02; //gyro sensitivity = 0.02V/deg/s
float GyY = raw_GyY * (5.0 / 32767.0) / 0.02;
float GyZ = raw_GyZ * (5.0 / 32767.0) / 0.02;


// Used for Debugging
Serial.print(AcX); Serial.print(" ");
Serial.print(AcY); Serial.print(" ");
Serial.print(AcZ); Serial.print(" ");
Serial.print(Tmp/340.00+36.53); Serial.print(" "); //equation for temperature in degrees C from
datasheet
Serial.print(GyX); Serial.print(" ");
Serial.print(GyY); Serial.print(" ");
Serial.println(GyZ); Serial.print(" ");


    delay(50);
}


// This is their code that I commented out.
// Wire.requestFrom(ADDR_SLAVE_I2C, PACKET_SIZE);
// byte data[PACKET_SIZE];
// int i = 0;
// while(Wire.available())
// {
//   data[i] = Wire.read();
//   i++;
// }
//
// // we use a c union to convert between byte[] and float

```

```

// union ROL_tag {byte b[4]; float fval;} ROL_Union;
// union PIT_tag {byte b[4]; float fval;} PIT_Union;
// union YAW_tag {byte b[4]; float fval;} YAW_Union;
//
// ROL_Union.b[0] = data[0];
// ROL_Union.b[1] = data[1];
// ROL_Union.b[2] = data[2];
// ROL_Union.b[3] = data[3];
// if(isnan(ROL_Union.fval) != 1)
// {
//   angleX = ROL_Union.fval;
// }
//
// PIT_Union.b[0] = data[4];
// PIT_Union.b[1] = data[5];
// PIT_Union.b[2] = data[6];
// PIT_Union.b[3] = data[7];
// if(isnan(PIT_Union.fval) != 1)
// {
//   angleY = PIT_Union.fval;
// }
//
// YAW_Union.b[0] = data[8];
// YAW_Union.b[1] = data[9];
// YAW_Union.b[2] = data[10];
// YAW_Union.b[3] = data[11];
// if(isnan(YAW_Union.fval) != 1)
// {
//   angleZ = YAW_Union.fval;
// }

```

// Each motor is named and attached to an Arduino pin
 // Here is how our motors are oriented:

```

//   M1(Front)
//   |
// M4---|---M2
//   |
//   M3(Back)

```

// Quadcopter is set up in "plus" orientation

Servo Motor1; //calls each motor as a "servo"

```
Servo Motor2;  
Servo Motor3;  
Servo Motor4;
```

```
void motors_initialize(){  
    Motor1.attach(PIN_MOTOR1); //assigns pins to motors from configuration.h file  
    Motor2.attach(PIN_MOTOR2);  
    Motor3.attach(PIN_MOTOR3);  
    Motor4.attach(PIN_MOTOR4);  
  
    Motor1.writeMicroseconds(MOTOR_ZERO_LEVEL); //sets all motors at 50%  
    Motor2.writeMicroseconds(MOTOR_ZERO_LEVEL);  
    Motor3.writeMicroseconds(MOTOR_ZERO_LEVEL);  
    Motor4.writeMicroseconds(MOTOR_ZERO_LEVEL);  
}
```

```
void motors_arm(){  
    Motor1.writeMicroseconds(MOTOR_ZERO_LEVEL);  
    Motor2.writeMicroseconds(MOTOR_ZERO_LEVEL);  
    Motor3.writeMicroseconds(MOTOR_ZERO_LEVEL);  
    Motor4.writeMicroseconds(MOTOR_ZERO_LEVEL);  
}
```

```
void update_motors(int M1, int M2, int M3, int M4)  
{  
    Motor1.writeMicroseconds(M1);  
    Motor2.writeMicroseconds(M2);  
    Motor3.writeMicroseconds(M3);  
    Motor4.writeMicroseconds(M4);  
}
```

```
PID roll_controller(&pid_roll_in, &pid_roll_out, &pid_roll_setpoint, 5.0, 0.0, 0.0,  
REVERSE);  
PID pitch_controller(&pid_pitch_in, &pid_pitch_out, &pid_pitch_setpoint, 5.0, 0.0, 0.0,  
REVERSE);  
PID yaw_controller(&pid_yaw_in, &pid_yaw_out, &pid_yaw_setpoint, 1.0, 0.0, 0.0,  
DIRECT);
```

```
void pid_initialize() {  
    roll_controller.SetOutputLimits(ROLL_PID_MIN,ROLL_PID_MAX);  
    pitch_controller.SetOutputLimits(PITCH_PID_MIN,PITCH_PID_MAX);  
    yaw_controller.SetOutputLimits(YAW_PID_MIN,YAW_PID_MAX);  
}
```



```

roll_controller.SetMode(AUTOMATIC);
pitch_controller.SetMode(AUTOMATIC);
yaw_controller.SetMode(AUTOMATIC);
roll_controller.SetSampleTime(10);
pitch_controller.SetSampleTime(10);
yaw_controller.SetSampleTime(10);
}

```

```

void pid_update(){
    pid_roll_in = angleX;
    pid_pitch_in = angleY;
    pid_yaw_in = angleZ;
}

```

```

void pid_compute() {
    roll_controller.Compute();
    pitch_controller.Compute();
    yaw_controller.Compute();
}

```

```

volatile int pwm_start_time[4];
uint8_t latest_interrupted_pin;

```

```

void rx_initialize() {
    pinMode(PIN_RX_ROLL, INPUT); digitalWrite(PIN_RX_ROLL, HIGH); //Channel 1 on
reciever
    PCintPort::attachInterrupt(PIN_RX_ROLL, &rising, RISING);

    pinMode(PIN_RX_PITCH, INPUT); digitalWrite(PIN_RX_PITCH, HIGH); //Channel 2 on
reciever
    PCintPort::attachInterrupt(PIN_RX_PITCH, &rising, RISING);

    pinMode(PIN_RX_THROTTLE, INPUT); digitalWrite(PIN_RX_THROTTLE, HIGH);
//Channel 3 on reciever
    PCintPort::attachInterrupt(PIN_RX_THROTTLE, &rising, RISING);

    pinMode(PIN_RX_YAW, INPUT); digitalWrite(PIN_RX_YAW, HIGH); //Channel 4 on
reciever
    PCintPort::attachInterrupt(PIN_RX_YAW, &rising, RISING);
}

```

```

void rising()

```

```

{
  latest_interrupted_pin=PCintPort::arduinoPin;
  PCintPort::attachInterrupt(latest_interrupted_pin, &falling, FALLING);
  pwm_start_time[latest_interrupted_pin-RX_PINS_OFFSET] = micros();
}

```

```

void falling() {
  latest_interrupted_pin=PCintPort::arduinoPin;
  PCintPort::attachInterrupt(latest_interrupted_pin, &rising, RISING);
  rx_values[latest_interrupted_pin-RX_PINS_OFFSET] = micros()-
  pwm_start_time[latest_interrupted_pin-RX_PINS_OFFSET];
}

```

```

void debug_process(){
#ifdef DEBUG_OUTPUT

#ifdef DEBUG_ANGLES
  Serial.print(F("X:"));
  Serial.print((float)(angleX));
  Serial.print('\t');
  Serial.print(F("Y:"));
  Serial.print((float)(angleY));
  Serial.print('\t');
  //Serial.print(F("Z:"));
  //Serial.print((float)(angleZ));
  //Serial.print('\t');
#endif
#endif

```

```

#ifdef DEBUG_RX
  Serial.print('\t');
  Serial.print(F("RX_Roll:"));
  Serial.print(rx_values[0]);
  Serial.print('\t');
  Serial.print(F("RX_Pitch:"));
  Serial.print(rx_values[1]);
  Serial.print('\t');
  Serial.print(F("RX_Throttle:"));
  Serial.print(rx_values[2]);
  Serial.print('\t');
  //Serial.print(F("RX_Yaw:"));
  //Serial.print(rx_values[3]);
  //Serial.print('\t');
#endif

```

```
#ifdef DEBUG_PID
    Serial.print(F("PID_R:"));
    Serial.print(pid_roll_in);Serial.print(',');
    Serial.print(pid_roll_setpoint);Serial.print(',');
    Serial.print(pid_roll_out);
    Serial.print('\t');
    Serial.print(F("PID_P:"));
    Serial.print(pid_pitch_in);Serial.print(',');
    Serial.print(pid_pitch_setpoint);Serial.print(',');
    Serial.print(pid_pitch_out);
    Serial.print('\t');
    //Serial.print(F("PID_Y:"));
    //Serial.print(pid_yaw_in);Serial.print(',');
    //Serial.print(pid_yaw_setpoint);Serial.print(',');
    //Serial.print(pid_yaw_out);
    //Serial.print('\t');
#endif
```

```
#ifdef DEBUG_MOTORS
    Serial.print('\t');
    Serial.print(F("M1:"));
    Serial.print(M1);
    Serial.print('\t');
    Serial.print(F("M2:"));
    Serial.print(M2);
    Serial.print('\t');
    Serial.print(F("M3:"));
    Serial.print(M3);
    Serial.print('\t');
    Serial.print(F("M4:"));
    Serial.print(M4);
    Serial.print('\t');
#endif
```

```
#ifdef DEBUG_LOOP_TIME
    Serial.print('\t');
    unsigned long elapsed_time = micros() - prev_time;
    Serial.print(F("Time:"));
    Serial.print((float)elapsed_time/1000);
    Serial.print(F("ms"));
    Serial.print('\t');
#endif
```

```
    Serial.println();  
#endif  
}
```

Automatic Dump Truck

ME 445: Final Project



Johnathan Rearick
Brandi Ritenour

18 December 2014

Table of Contents

Acknowledgements.....	3
Introduction	4
Bill of Materials	4
Hardware Components.....	4
4x4 RC Truck.....	4
Dump Truck Bed.....	5
SparkFun RedBot Main Board.....	5
Lego Linear Actuator and Motor.....	5
Sharp IR Distance Sensors.....	5
Load Cell.....	6
Adafruit Accelerometer	6
Linear Potentiometer	6
Adafruit ADS1115 16-bit ADC	6
Technical Approach.....	6
Discussion of Results.....	7
Conclusions and Recommendations.....	8
Appendices.....	9
Appendix A: Arduino Source Code.....	9
Appendix B: Data Sheets & Calibration.....	14

Acknowledgements

We would like to thank Dr. Sommer for the knowledge and experience that we have gained throughout the semester. We did not have much knowledge about electronics coming into this course, and was able to teach us a lot. We were also greatly appreciative of the help we received from Mike Robinson. He was of great assistance with regards to trouble shooting any problems we were having with code, hardware, or any general questions. This course has provided us with the learning experiences that we believe will be able to be taken with us throughout our future academic and professional careers.

Introduction

The team constructed an automatic dump truck that would move to a location and dump a load by itself. This objective of this project was to build, debug, and demonstrate materials learned throughout the course. The goal was for the project to cost about \$100-\$150, but be complex enough to learn something new and interesting. The robot combined hardware and software using Arduino, sensors, actuators, servos, and motors.

A dump truck was constructed by attaching a bed off of a toy Tonka truck to an existing four wheel drive RC truck frame. Also included in the mounting of the bed was a hinge, linear actuator, accelerometer, and load cell. Distance sensors were mounted on the frame as well to measure the distance behind the vehicle.

The goal of this truck was that when there is a sufficient load in the bed, representing a “full” dump truck, to drive to a location of certain distance, and dump the load. Specifically, the truck backed up 12” from a wall, which is figured to represent an existing pile of dirt.

Bill of Materials

Item	Quantity	Supplier	Cost
4x4 RC Truck	1	Dr. Brennan’s Lab	\$0
Tonka Dump Bed	1	Walmart	\$24.95
RedBot Main Board	1	SparkFun	\$49.95
Lego Linear Actuator	1	445 Lab	\$0
Lego Motor	1	445 Lab	\$0
Sharp IR Distance Sensors	2	445 Lab	\$0
Load Cell	1	Amazon	\$9.23
Adafruit Accelerometer	1	Adafruit	\$14.95
Adafruit ADS1115 16-bit	1	Adafruit	\$14.95
Miscellaneous Hardware (screws, nuts, washers, etc)	N/A	Lowe’s	~\$10.00

The total cost of this project came out to be \$124.03. The team was able to keep the cost down by trying to utilize materials that were already available in the lab. The next section of this report will describe the materials that were used for this project in more detail.

Hardware Components

The following sub-titles represent all of the individual components that were combined in the making of this dump truck. Individually, a lot of time was spent to get each piece working, as well as about equal combined time to combine them all in this autonomous vehicle.

4x4 RC Truck

A four wheel drive RC truck was received from the advanced Mechatronics lab with help from Mike Robinson. The cost and manufacturer are unknown to the group, but expected to cost around \$80 from comparison. This truck consisted of a servo to control the steering, as well as a servo to control the four drive wheels. Due to the extra needed power for the drive wheels, it has a built in battery. The next important component was the dump truck bed.

Dump Truck Bed

The team purchased a toy Tonka truck from a local store, and tore it apart. The bed was attached using a hinge and a linear actuator. A load cell was mounted on the third actuated point to measure the weight in the bed. Also, the “cab” of the truck was used to go on the truck for demonstration purposes. On the back of the bed, two distance sensors were mounted on a piece of aluminum angle iron.

SparkFun RedBot Main Board

The project started by using an Arduino Uno micro-processor. However, Mike introduced the team to this micro-processor which created a neater wiring job. Using the Arduino Uno, a separate bread board was necessary to split the power supplies for the sensors, as well as a separate motor driver for the dump actuator. The RedBot main board is convenient due to having groups of three-pin inputs, which all included a ground and voltage supply pin. This created a much neater wiring job, lowering worries of wires coming apart. It had analog pins meant for sensors, as well as servo sets, using PWM. The built-in motor drivers worked very well, only having to connect the two motor wires directly. A library from sparkfun had to be included on the computer, and called in the code to operate these drivers.

The board, as well as the Arduino, include a place to attach an external voltage supply. The first idea was using a single 9 volt battery, but a 6 volt supply was accomplished using 4 AA batteries. The 6 volt supply was much better suited to work along with the 5 volt supply of the main board. Also, the drive servo, as stated in the previous section, has a 7.2 volt battery, which would fight with the internal 5 volts. This was prevented by by-passing the 5 volt connection between the board and the servo. However, the grounds of all components were connected. The data sheet is found in figure 4, appendix B.

Lego Linear Actuator and Motor

This linear actuator was used to tilt the dump truck bed when the load needed to be emptied. Instead of using pneumatic or hydraulic cylinders like most linear actuators, this style used a Lego DC motor. The main reason is that this needs the fewest components, with the least mess. Hydraulic actuators first need a bulky pump, however, another problem relies on what happens when a leak becomes present, getting into the other electronics. Pneumatic also was not readily available since a primary needs for compressing the air was not feasible. This also produces air leaks and bulky pumps. This Lego actuator uses a screw design to extend the cylinder.

Sharp IR Distance Sensors

These IR distance sensors were used to measure the distance to the wall behind the vehicle. They produced a voltage that was proportional to a distance. To calibrate this, an excel spreadsheet was used. One column represented the distance measured with a tape measure, and the other column held the measured voltage. A fit line was produced, giving a conversion equation to put into the code. This chart can be viewed in appendix B, figure 1. The sensors, however, produced a lot of problems in this project. They did not produce a consistent output. Due to this inconsistency, troubles were produced within the controlling equations. The data sheet is found in figure 5, appendix B.

Load Cell

A load cell was mounted between the linear actuator and the dump bed to measure the weight inside of the bed. This weight was used to control when the truck moves to a new location. Precision on this process was not as good as desired, but sufficient for this project. The position of the bed caused extreme differences in the readings, especially when near the pre-existing support under the bed. Also, the flex in the bed produces different results, as well as the movement in the actuator/motor set-up. These inefficiencies were dealt with by using a lower minimum weight than the desired. If more precision was needed, it would have to be mounted more securely, and probably amplified more.

Adafruit Accelerometer

An AXDL335 accelerometer, by Adafruit, was initially used in this project to measure the angle of the bed. This was necessary to control how far to control the linear actuator, as well as finding the horizontal position. To do this, the base acceleration values had to be found, and subtracted from the voltage output. Now, the read voltage could be directly used as being proportional. The accelerometer was mounted in the x-y plane, so $\tan^{-1}(y/x)$ was used to measure the angle. The data sheet is found in figure 3, appendix B. However, there was an extreme amount of noise that could not be solved with the filter that was used, so this idea got changed to a linear potentiometer.

Linear Potentiometer

After deciding to forget about the accelerometer to measure an angle, the group turned to a linear potentiometer to measure the displacement as the bed lowers. It was used to stop the bed at a certain position. To control the bed motion upwards, it was timed to a specific length. On the way down, it lands onto the potentiometer, stopping at the same location each time. This was necessary to start the bed in the same position each time due to the load cell reading. If it starts in a different location, the sensitive load cell will always read differently, making it hard to control properly.

Adafruit ADS1115 16-bit ADC

This analog-digital converter supports 16 bits of resolution where the Arduino only is capable of 10 bits. The main reason that this chip was used, however, was to measure differential voltages with amplification from the load cell. This chip uses I²C and has four inputs that can either measure four different voltages at a time, or two different differential voltages. The problem with measuring these differential voltages is that the measurements are extremely small. This chip allows amplification up to 16 times, with an accuracy of 0.256V. This chip worked extremely well to easily measure the output of the load cell, however, there are more precise ways that this could have been done if necessary. The data sheet is found in figure 2, in appendix B.

Technical Approach

To make this robot, all of the previous components were put together in hardware and software. Reactive controlling is the mainframe of the controller. To start, each individual piece was studied, and software was produced to control them by themselves. Then, all of the hardware was mounted onto the RC frame. The bed was able to be mounted onto the truck by rail extensions, without drilling any holes into it. The team did not want to apply unnecessary stresses, however, more importantly, it was out of respect for the borrowed equipment. Chicago type screws attached the bed onto a formed bar, acting as a hinge. Then, the load cell was mounted towards the front of the bed, with intent of attaching the linear actuator to the other end. A bracket was made to hold the dowel pin that

went through the actuator end. Next, the other end of the actuator had to be connected to the Lego motor. This motor had to be mounted in a proper position and way to allow tilting as the bed rises. Next, wiring was accomplished, including the ADS 1115 chip. This amplifying ADC took the two load cell outputs as inputs, a 5 volt supply from the main board, ground, and the output to the mainboard. The team put a header onto the chip, and male square connectors (.1"), rather than a breadboard or direct soldering. The distance sensors received direct connection of 5 volts and ground from the main board, as well as the output connections. Power and ground were also supplied to the load cell, as well as the differential output, from the mainboard. Finally, the servos were attached on separate pins, as well as the motor connections for the linear actuator. The built-in dual motor driver allowed easy wiring and simple operation. The Arduino code was then put together and debugged. This is available in Appendix A with sufficient commenting.

This project has been designed to work fairly well, however, the level of sensor precision is not very high. For example, the distance sensors work for this project, but due to the inconsistencies leave a hefty percentage of error from the physical distance. So, the distance from the wall is not perfectly accurate or precise. Also, there was a lot of noise produced in the angle measurement. So, a moving average filter was applied, but produced a lot of error still.

Discussion of Results

The team was faced with many challenges along this path. To start, libraries had to be added to the computer to operate the RedBot board and the Adafruit ADS 1115. This presented a problem because they were saved in the wrong locations. Then, the correct location was discovered (Arduino-Libraries), however the files did not get un-zipped properly. Once these libraries were applied correctly, the RedBot mainboard motor drivers were easy to use with just one line of code. The sign on the duty cycle directly controlled the desired direction as well. This eliminated the need for 5 wires as used previously in the motor drivers, as well as more lines of code. Next, the differential voltage from the load cell required an installed library to use the ADS 1115 chip.

Another hardware problem that produced constant challenges came from the RC servos. These servos had severe inconsistencies, and a lot of the time would not work. The team had to constantly reset them, try different codes, and ensure the batteries were good. The reason for this problem never was determined, however, the group feels that it is a hardware problem, possibly partially fried.

The plastic linear actuator/motor combination produced a major problem due to movement. As the truck accelerated, which was pretty quick, the bed would move up and down. This produced problems mainly to the weight readings and also to the distance and acceleration readings. When bouncing, the severe change in load could make the truck stop abruptly until the weight settles, which produces a horrible bucking effect. To help this, the needed load was lowered below the applied load to attempt to compensate for this.

A link to the presented video is followed.

http://youtu.be/zPWCiT_fXg4

Conclusions and Recommendations

To conclude, the team was able to take away several new skills from the experience of completing this project. Since neither of the group members had hardly any experience with programming or electronics prior to this course, there was a lot of important knowledge that was gained. For example, the team had to learn how the Arduino Uno worked and how to code with it. Even though it took a while to catch on, both members finally got the hang of it and can agree that it was a very valuable skill to take away. Debugging the code still has room for improvement for the team, and it can be concluded that it is just something that will improve with practice. Further, the opportunity was taken to discover the differences between the Arduino Uno and the SparkFun RedBot main board, as well as the advantages and limitations of each. Also, the team was able to learn and experiment with PD control. A better understanding was acquired of how both proportional and derivative gains affect a system. Additionally, both team members were able to keep working on basic professional skills such as teamwork, communication, organization, and time management. Asking questions while still balancing how to learn on one's own is another essential tool that was acquired. All of these skills are of great value and can be taken into any professional work environment.

If this project were to be redone, there are several things that the team would reconsider. First, the dump bed could be mounted differently onto the truck. The way that it is now is seemingly not as stable as it could be. The bed tends to move as the car moves, altering the load cell and distance sensor readings. Adding more hardware to further secure the bed could help this problem. The Lego pieces holding the linear actuator/motor together produce these unnecessary movements.

The team would also choose different distance sensors. The sensors that are currently used were chosen due to availability in the lab. They are fairly inexpensive, making them inconsistent and providing narrow reading ranges. This limitation caused it to be very difficult to accurately calibrate the sensors. Originally, the team wanted to make the truck follow a path along a wall and avoid obstacles; however, to do this, more accurate sensors would certainly be necessary. The overall project could have been easier and more accurate if more expensive sensors with greater consistency and a wider range of readings were used.

Appendices

Appendix A: Arduino Source Code

// The following code is used as an ME project. A dump will back up to a pile only when there is sufficient weight.

// Then, it will dump the load. It uses PD control.

// The following code is used as an ME project. A dump will back up to a pile only when there is sufficient weight.

// Then, it will dump the load. It uses PD control.

```
#include <Servo.h> // This is used to attach the servos.
```

```
#include <Wire.h>
```

```
#include <Adafruit_ADS1015.h> // This includes the adafruit library to use the ADS 1115.
```

```
#include "RedBot.h" // This includes the library to use the RedBot motor drivers.
```

```
Adafruit_ADS1115 ads;
```

```
RedBotMotor motor;
```

```
float RRsensor = 6;
```

```
float LRSensor = 7;
```

```
Servo myservoMove;
```

```
Servo myservoSteer;
```

```
float pastTime = 0;
```

```
float pastDistance = 0;
```

```
float DriveDutyCycle;
```

```
float GainD = 0.5;
```

```
float GainP = 30;
```

```
float GoalDistance = 12;
```

```
float timeAtDump = 0;
```

```
float N = 0.99;
```

```
float pastWeight = 0;
```

```
float pastVoltageInput = 0;
```

```

void setup() {
  // setup code here, to run once:
  pinMode (6, OUTPUT);
  pinMode (7, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  Serial.begin(9600);
  myservoMove.attach(10);
  myservoSteer.attach(9);
  ads.setGain(GAIN_SIXTEEN); // 16x gain +/- 0.256V 1 bit = 0.0078125mV
  ads.begin();
}

void loop() {
  // main code here, to run repeatedly:
  // The following section is to read the load cell using the ADS 1115.
  float multiplier = 0.1875F;
  float results = ads.readADC_Differential_0_1();
  float LoadCellReading = multiplier * results;
  float WeightInTruck = -25.225*LoadCellReading - 186.63; // Conversion from voltage to weight.
  pastWeight = N*pastWeight + (1-N)*WeightInTruck;
  Serial.print(WeightInTruck);
  Serial.print(" ");

  // The following measures the distance behind the truck and converts to inches.
  float LRsensordReading = analogRead(LRsensord);
  float RRsensordReading = analogRead(RRsensord);
  float DistanceError = RRsensordReading - LRsensordReading;
  float DistanceInInches = 3266.6*pow(LRsensordReading,-.948);

```

```

Serial.print(LRsensorReading);

Serial.print(" ");

Serial.print(DistanceInInches);


// The following straightens the wheels and activates the drive servo initiation.
myservoSteer.write(90);
myservoMove.write(90);


// The following piece of code implements the PD control in the system.
float timeCounter = millis();
float curTime = timeCounter;
float timeChange = (curTime - pastTime)/1000;
float DerDriveCont = ((DistanceInInches - pastDistance)/(timeChange))*GainD;
float PropDriveCont = (DistanceInInches - GoalDistance)*GainP;
float VoltageInput = DerDriveCont + PropDriveCont;
pastVoltageInput = N*pastVoltageInput + (1-N)*VoltageInput;
// Serial.print(" ");
// Serial.println(VoltageInput);

pastTime = curTime;
pastDistance = DistanceInInches;


//The following activates the drive duty cycle and converts to usable signals.
if(VoltageInput > 0)
{
    DriveDutyCycle = abs(pastVoltageInput+90);
    if(DriveDutyCycle > 180.0)
    {
        DriveDutyCycle = 180;
    }
}

```

```

}
if(VoltageInput < 0)
{
    DriveDutyCycle = abs(pastVoltageInput-20);
    if(DriveDutyCycle < 0)
    {
        DriveDutyCycle = 0;
    }
}
Serial.print(" ");
Serial.print(DriveDutyCycle);

// Operates the drive servos only when sufficient weight is in the truck.
if (pastWeight > 80)
{
    myservoMove.write(DriveDutyCycle);
}
if(pastWeight < 80)
{
    myservoMove.write(90);
}

// Dumps the load when close enough to the wall.
if(DistanceInInches <= 12.0)
{
    if(timeAtDump == 0)
    {
        timeAtDump = timeCounter;
    }
}

```



```
myservoMove.write(90);

float linearPotReading = analogRead(A1);

float DumpTime = 10000 + timeAtDump;

if(timeCounter < DumpTime)
{
    motor.rightDrive(255);
}

if(timeCounter >= DumpTime && linearPotReading < 100)
{
    motor.rightDrive(-255);
}

if(timeCounter > DumpTime && linearPotReading > 100)
{
    motor.rightDrive(0);
}

Serial.print(" ");

Serial.println(timeAtDump);

}
```

Appendix B: Data Sheets & Calibration

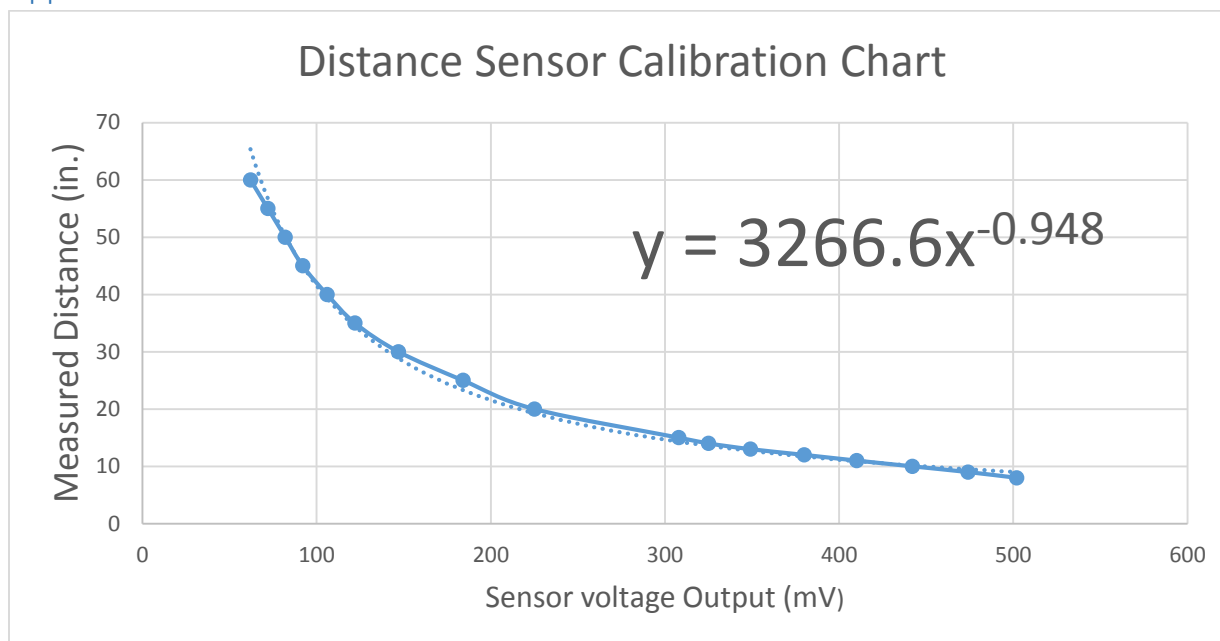


Figure 1



ADS1113
ADS1114
ADS1115

www.ti.com

SBAS444B –MAY 2009–REVISED OCTOBER 2009

Ultra-Small, Low-Power, 16-Bit Analog-to-Digital Converter with Internal Reference

Check for Samples: [ADS1113](#) [ADS1114](#) [ADS1115](#)

FEATURES

- **ULTRA-SMALL QFN PACKAGE:**
2mm × 1.5mm × 0.4mm
- **WIDE SUPPLY RANGE:** 2.0V to 5.5V
- **LOW CURRENT CONSUMPTION:**
Continuous Mode: Only 150µA
Single-Shot Mode: Auto Shut-Down
- **PROGRAMMABLE DATA RATE:**
8SPS to 860SPS
- **INTERNAL LOW-DRIFT
VOLTAGE REFERENCE**
- **INTERNAL OSCILLATOR**
- **INTERNAL PGA**
- **I²C™ INTERFACE:** Pin-Selectable Addresses
- **FOUR SINGLE-ENDED OR TWO
DIFFERENTIAL INPUTS (ADS1115)**
- **PROGRAMMABLE COMPARATOR
(ADS1114 and ADS1115)**

APPLICATIONS

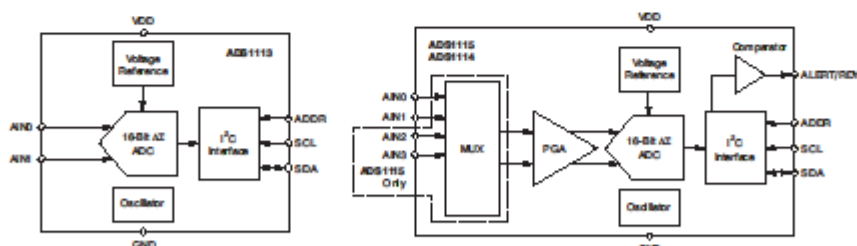
- **PORTABLE INSTRUMENTATION**
- **CONSUMER GOODS**
- **BATTERY MONITORING**
- **TEMPERATURE MEASUREMENT**
- **FACTORY AUTOMATION AND PROCESS
CONTROLS**

DESCRIPTION

The ADS1113, ADS1114, and ADS1115 are precision analog-to-digital converters (ADCs) with 16 bits of resolution offered in an ultra-small, leadless QFN-10 package or an MSOP-10 package. The ADS1113/4/5 are designed with precision, power, and ease of implementation in mind. The ADS1113/4/5 feature an onboard reference and oscillator. Data are transferred via an I²C-compatible serial interface; four I²C slave addresses can be selected. The ADS1113/4/5 operate from a single power supply ranging from 2.0V to 5.5V.

The ADS1113/4/5 can perform conversions at rates up to 860 samples per second (SPS). An onboard PGA is available on the ADS1114 and ADS1115 that offers input ranges from the supply to as low as ±256mV, allowing both large and small signals to be measured with high resolution. The ADS1115 also features an input multiplexer (MUX) that provides two differential or four single-ended inputs.

The ADS1113/4/5 operate either in continuous conversion mode or a single-shot mode that automatically powers down after a conversion and greatly reduces current consumption during idle periods. The ADS1113/4/5 are specified from -40°C to +125°C.



Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

I²C is a trademark of NXP Semiconductors.

All other trademarks are the property of their respective owners.

PRODUCTION DATA information is current as of publication date. Products conform to specifications per the terms of the Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

Copyright © 2009, Texas Instruments Incorporated

Figure 2: ADS 1115 Data Sheet



Small, Low Power, 3-Axis $\pm 3g$ Accelerometer

ADXL335

FEATURES

3-axis sensing
 Small, low profile package
 4 mm × 4 mm × 1.45 mm LFCSP
 Low power: 350 μ A (typical)
 Single-supply operation: 1.8 V to 3.6 V
 10,000 g shock survival
 Excellent temperature stability
 BW adjustment with a single capacitor per axis
 RoHS/WEEE lead-free compliant

APPLICATIONS

Cost sensitive, low power, motion- and tilt-sensing applications
 Mobile devices
 Gaming systems
 Disk drive protection
 Image stabilization
 Sports and health devices

GENERAL DESCRIPTION

The ADXL335 is a small, thin, low power, complete 3-axis accelerometer with signal conditioned voltage outputs. The product measures acceleration with a minimum full-scale range of $\pm 3g$. It can measure the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion, shock, or vibration.

The user selects the bandwidth of the accelerometer using the C_x , C_y , and C_z capacitors at the X_{OUT} , Y_{OUT} , and Z_{OUT} pins. Bandwidths can be selected to suit the application, with a range of 0.5 Hz to 1600 Hz for the X and Y axes, and a range of 0.5 Hz to 550 Hz for the Z axis.

The ADXL335 is available in a small, low profile, 4 mm × 4 mm × 1.45 mm, 16-lead, plastic lead frame chip scale package (LFCSP_LQ).

FUNCTIONAL BLOCK DIAGRAM

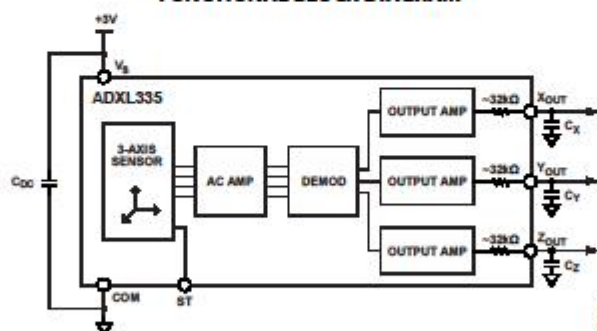


Figure 1.

Rev. B

Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use, nor for any infringements of patents or other rights of third parties that may result from its use. Specifications subject to change without notice. No license is granted by implication or otherwise under any patent or patent rights of Analog Devices. Trademarks and registered trademarks are the property of their respective owners.

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
 Tel: 781.329.4700 www.analog.com
 Fax: 781.461.3113 ©2009–2010 Analog Devices, Inc. All rights reserved.

Figure 3: ADXL 335

Features

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
 - 131 Powerful Instructions – Most Single Clock Cycle Execution
 - 32 x 8 General Purpose Working Registers
 - Fully Static Operation
 - Up to 20 MIPS Throughput at 20 MHz
 - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
 - 4/8/16/32K Bytes of In-System Self-Programmable Flash program memory (ATmega48PA/88PA/168PA/328P)
 - 256/512/512/1K Bytes EEPROM (ATmega48PA/88PA/168PA/328P)
 - 512/1K/1K/2K Bytes Internal SRAM (ATmega48PA/88PA/168PA/328P)
 - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
 - Data retention: 20 years at 85°C/100 years at 25°C⁽¹⁾
 - Optional Boot Code Section with Independent Lock Bits
 - In-System Programming by On-chip Boot Program
 - True Read-While-Write Operation
 - Programming Lock for Software Security
- Peripheral Features
 - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
 - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
 - Real Time Counter with Separate Oscillator
 - Six PWM Channels
 - 8-channel 10-bit ADC in TQFP and QFN/MLF package
 - Temperature Measurement
 - 6-channel 10-bit ADC in PDIP Package
 - Temperature Measurement
 - Programmable Serial USART
 - Master/Slave SPI Serial Interface
 - Byte-oriented 2-wire Serial Interface (Philips I²C compatible)
 - Programmable Watchdog Timer with Separate On-chip Oscillator
 - On-chip Analog Comparator
 - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
 - Power-on Reset and Programmable Brown-out Detection
 - Internal Calibrated Oscillator
 - External and Internal Interrupt Sources
 - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
 - 23 Programmable I/O Lines
 - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
 - 1.8 - 5.5V for ATmega48PA/88PA/168PA/328P
- Temperature Range:
 - -40°C to 85°C
- Speed Grade:
 - 0 - 20 MHz @ 1.8 - 5.5V
- Low Power Consumption at 1 MHz, 1.8V, 25°C for ATmega48PA/88PA/168PA/328P:
 - Active Mode: 0.2 mA
 - Power-down Mode: 0.1 µA
 - Power-save Mode: 0.75 µA (Including 32 kHz RTC)



**8-bit AVR®
Microcontroller
with 4/8/16/32K
Bytes In-System
Programmable
Flash**

**ATmega48PA
ATmega88PA
ATmega168PA
ATmega328P**

Rev. 8161D-AVR-10/09



Figure 4: RedBot Mainboard

GP2Y0A02YK

Long Distance Measuring Sensor

■ Features

1. Less influence on the colors of reflected objects and their reflectivity, due to optical triangle measuring method
2. Distance output type
(Detection range: 20 to 150cm)
3. An external control circuit is not necessary
Output can be connected directly to a microcomputer

■ Applications

1. For detection of human body and various types of objects in home appliances, OA equipment, etc

■ Absolute Maximum Ratings (T_a=25°C)

Parameter	Symbol	Rating	Unit
Supply voltage	V_{CC}	-0.3 to +7	V
*1 Output terminal voltage	V_O	-0.3 to $V_{CC}+0.3$	V
Operating temperature	T_{opr}	-10 to +60	°C
Storage temperature	T_{str}	-40 to +70	°C

*1 Open collector output

■ Recommended Operating Conditions

Parameter	Symbol	Rating	Unit
Operating Supply voltage	V_{CC}	4.5 to 5.5	V

■ Outline Dimensions

(Unit : mm)

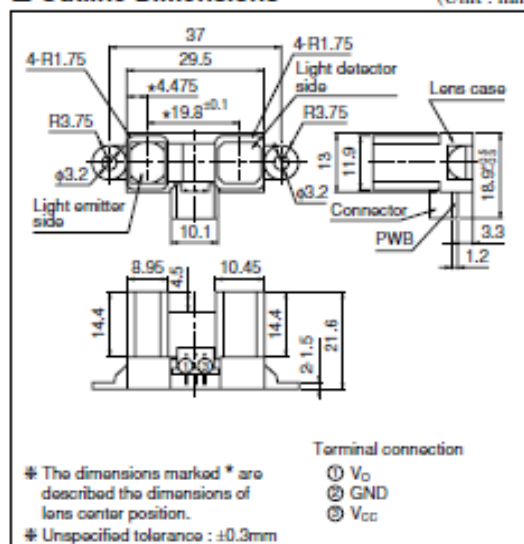


Figure 5: IR Distance Sensors